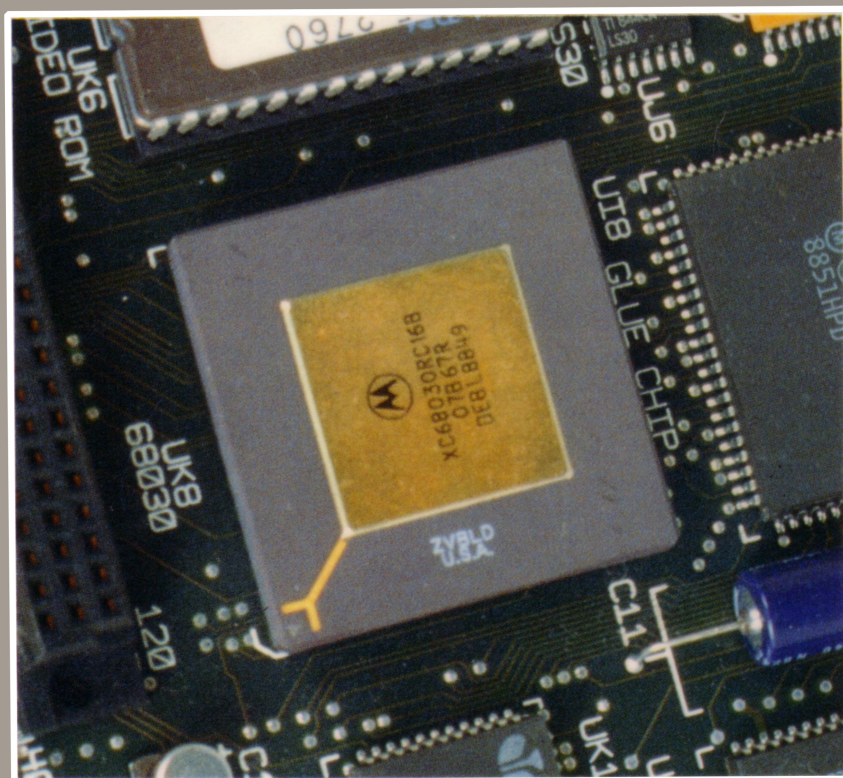
MICROPROCESSORI

68020
68030

PROGRAMMAZIONE INTERFACCIAMENTO E PROGETTAZIONE

THOMAS L. HARMAN

GRUPPO EDITORIALE JACKSON

Prentice Hall International

68020
68030

PROGRAMMAZIONE INTERFACCIAMENTO E PROGETTAZIONE

THOMAS L. HARMAN

GRUPPO EDITORIALE JACKSON

Prentice Hall International

INDICE

PRESENTAZIONE DELL'EDIZIONE ITALIANA

L'MC68020 e l'MC68030 sono i principali membri della nuova famiglia di microprocessori VLSI a 32 bit prodotti dalla Motorola, che offrono una potente capacità di supporto per applicazioni su grandi e piccoli sistemi. Ciascuno di essi è un microprocessore realizzato su di un singolo chip, progettato per fungere da unità centrale di elaborazione in un sistema avanzato di computer. Questo libro descrive in dettaglio le loro peculiarità ed i possibili impieghi, per quanto concerne la programmazione in linguaggio assembler, la progettazione di interfacce e di sistemi e la realizzazione di prodotti.

Oltre ad essere utilizzabile semplicemente come una fonte di consultazione da parte dei programmatori addetti allo sviluppo del software — in quanto gli argomenti sono organizzati in base alla loro funzione ed all'importanza che rivestono per il progetto di programmi, di interfacce o di sistemi — in effetti questo volume risulta interessante anche dal punto di vista didattico, in quanto si tratta di un libro di testo adottato in alcune università statunitensi. Una delle sue finalità primarie è quella di fornire allo studente o al professionista di computer tutte le informazioni sugli aspetti significativi della progettazione di un sistema basato sull'MC68020 o sull'MC68030.

Lo studio di questo libro condurrà quindi il lettore ad acquisire la padronanza dei sofisticati "strumenti" software costituiti dalle istruzioni dei microprocessori MC68020/MC68030, nonché le conoscenze necessarie per poter utilizzare al meglio tali strumenti e quindi ridurre sia i tempi che i costi per lo sviluppo di un determinato prodotto.

Marcello G. Falconi

PREFAZIONE

L'introduzione sul mercato della famiglia di microprocessori MC68020 ha condotto ad una nuova generazione di processori a 32 bit. I principali membri di questa famiglia sono l'MC68020 e l'MC68030. Ciascuno di essi è un microprocessore su singolo chip progettato per operare come unità centrale di elaborazione di un sistema di computer avanzato. Le caratteristiche e gli usi dei processori della Motorola sono trattati in dettaglio in questo libro. Di tali processori sono evidenziati gli aspetti che riguardano la programmazione in linguaggio assembler, la progettazione dell'interfaccia e la progettazione del sistema.

Uno degli scopi più importanti di questo volume è quello di fornire allo studente o al professionista di computer tutte le informazioni sugli aspetti significativi della progettazione di un sistema basato sull'MC68020. Inoltre, il libro può servire come fonte di consultazione, in quanto gli argomenti sono organizzati in base alla loro funzione ed all'importanza che rivestono per il progetto di programmi, di interfacce o di sistemi.

Il volume è organizzato in cinque parti, come indicato nelle "Descrizioni dei capitoli" in questa prefazione. I primi quattro capitoli presentano al lettore la famiglia dell'MC68020. Questi capitoli contengono anche un'introduzione ai microcomputer ed all'aritmetica dei computer. I capitoli dal 5 al 9 trattano le tecniche di programmazione in linguaggio assembler. I capitoli dal 10 al 12 riguardano la progettazione e lo sviluppo di sistemi per computer basati sull'MC68020. Dal cap. 13 al cap. 14 sono trattati gli aspetti dell'hardware dell'MC68020, incluso il VMEbus. Il cap. 15 descrive il processore MC68030. Prima delle appendici, il lettore potrà trovare le risposte alla maggior parte dei problemi presentati nel testo. Le appendici costituiscono un utile riepilogo per il programmatore, in quanto comprendono il linguaggio assembler ed il linguaggio-macchina per la famiglia dell'MC68020. Per concludere, alla fine del volume ci sono un indice delle istruzioni ed un indice analitico.

Nelle prossime pagine saranno presentati un sommario dei principali argomenti trattati nel libro, una lista di acronimi, un elenco di programmi per argomento ed una lista di ulteriori riferimenti bibliografici che potrebbero essere utilmente impiegati come supplemento a questo libro.

DESCRIZIONI DEI CAPITOLI

I capitoli dall'1 al 4 presentano al lettore la famiglia di prodotti dell'MC68020, i microcomputer, i fondamenti dell'aritmetica di macchina, e l'unità di elaborazione centrale dell'MC68020.

- Il *capitolo 1* presenta un certo numero di applicazioni del microprocessore MC68020 della Motorola. È descritto anche il supporto hardware e software per la famiglia di dispositivi a circuiti integrati dell'MC68020.

- Il *capitolo 2* discute l'organizzazione di sistemi tipici di microcomputer. È descritta la funzione dei principali componenti del sistema (CPU, memoria, ingresso/uscita) e viene posta in rilievo l'importanza della lunghezza di word della CPU e dell'intervallo d'indirizzamento. Sono forniti tre distinti punti di vista del sistema: quello del progettista di sistema, quello del programmatore in linguaggio assembler, e quello del progettista d'interfaccia.

- Il *capitolo 3* spiega la rappresentazione interna di numeri e di caratteri del processore MC68020. Insieme coi dettagli delle operazioni aritmetiche, sono trattate le notazioni binaria, decimale, e di virgola mobile. Viene presentato anche il codice ASCII per i caratteri alfanumerici.

- Il *capitolo 4* è dedicato ad una discussione delle caratteristiche del processore MC68020, che viene introdotto dapprima come un chip di circuiti integrati, prima che siano presentate le sua caratteristiche come processore programmabile. È trattata anche l'organizzazione della memoria in un tipico sistema basato sull'MC68020.

I capitoli dal 5 al 9 sono dedicati alle tecniche di programmazione che impiegano l'MC68020. Per illustrare le numerose capacità dell'MC68020, si è usato il linguaggio assembler del processore.

- Il *capitolo 5* introduce il linguaggio assembler dell'MC68020. Gli argomenti trattati sono lo sviluppo del software, le caratteristiche del linguaggio e le varie modalità d'indirizzamento dell'MC68020.

- Il *capitolo 6* presenta tre importanti categorie di istruzioni per l'MC68020. Sono discusse le istruzioni per il trasferimento di dati, per il controllo del programma e le subroutine.

- Il *capitolo 7* contiene spiegazioni ed esempi di programmi concernenti le capacità aritmetiche dell'MC68020. Sono presentate l'aritmetica binaria, l'aritmetica decimale e le conversioni tra codici ASCII, binari e BCD. Sono presentate anche le tecniche di trasferimento di I/O.

- Il *capitolo 8* è un'introduzione alle istruzioni logiche, di scorrimento e rotazione e di manipolazione di bit.

- Il *capitolo 9* completa lo studio delle tecniche di programmazione fondamentali. Sono trattati i metodi per la creazione di codice indipendente dalla posizione. Sono forniti esempi di programmi per la manipolazione di strutture di dati, inclusi array e liste. Sono presentati alcuni metodi più avanzati per le subroutine.

I capp. 10, 11 e 12 sono dedicati ad alcuni aspetti dell'MC68020 che determinano l'attività del sistema del computer. I capp. 10 e 11 sono d'interesse per il progettista di sistema e per il programmatore che crea programmi di supervisore. Il cap. 12 tratta alcune applicazioni avanzate dell'MC68020.

- Il *capitolo 10* considera i vari stati e modi operativi del processore. Sono trattate le istruzioni del linguaggio assembler per il controllo del processore e sono forniti alcuni esempi di procedure d'inizializzazione.

- Il *capitolo 11* discute l'elaborazione delle eccezioni. Queste comprendono interruzioni, trappole e varie condizioni di errore riconosciute dalla CPU durante l'esecuzione del programma.

- Il *capitolo 12* introduce varie caratteristiche avanzate dell'MC68020. Sono descritte le memoria cache e l'interfaccia di coprocessore. Sono presentate le tecniche di programmazione per i coprocessori MC68851 e MC68881. Viene discusso lo stack principale e vengono svolte considerazioni di multielaborazione.

I capp. 13 e 14 trattano gli aspetti dell'hardware dell'MC68020 e di sistemi basati sull'MC68020.

- Il *capitolo 13* presenta le tecniche di programmazione dei chip periferici ed i requisiti d'interfacciamento dell'MC68020.

- Il *capitolo 14* descrive le caratteristiche e le applicazioni del VMEbus. Sono trattate la progettazione e la selezione del sistema per sistemi di VMEbus.

Il cap. 15 descrive il microprocessore MC68030.

- Il *capitolo 15* presenta le affinità e le differenze tra l'MC68020 e l'MC68030. Sono spiegati i vantaggi dell'MC68030 rispetto all'MC68020.

Sommario degli argomenti principali

LISTA DI ACRONIMI

ACIA *Asynchronous Communication Interface Adapter*
(adattatore d'interfaccia per comunicazioni asincrone)

ASCII *American Standard Code for Information Interchange*
(codice americano standard per lo scambio di informazioni)

ALU *Arithmetic and Logic Unit*
(unità aritmetico/logica)

BCD *Binary Coded Decimal*
(decimale codificato in binario)

CAE *Computer Aided Engineering*
(ingegnerizzazione assistita dal computer)

CPU *Central Processing Unit*
(unità centrale di elaborazione)

CRT *Cathode Ray Tube*
(tubo a raggi catodici)

DIP *Dual In-line Package*
(contenitore duale in linea)

DMA *Direct Memory Access*
(accesso diretto alla memoria)

I/O *Input/Output*
(ingresso/uscita)

MMU *Memory Management Unit*
(unità di gestione della memoria)

MOS *Metal-Oxide Semiconductor*
(metallo-ossido-semiconduttore)

PIA *Peripheral Interface Adapter*
(adattatore d'interfaccia periferica)

PLA *Programmed Logic Array*
(matrice logica programmabile)

PTM *Programmed Timer Module*
(modulo di timer programmabile)

RAM *Random Access Memory*
(memoria ad accesso casuale)

ROM *Read Only Memory*
(memoria a sola lettura)

SSI *Small Scale Integration*
(integrazione su piccola scala)

VLSI *Very Large Scale Integration*
(integrazione su larghissima scala)

LISTA DI PROGRAMMI PER ARGOMENTO

Aritmetica decimale	
Addizione BCD di sei cifre	Paragrafo 7.5
Aritmetica in virgola mobile	
Tipi di dati	Paragrafo 12.3
Analisi di Fourier	Paragrafo 12.3
Strutture di dati	
Confronto di stringhe di byte	Paragrafo 6.2
Tabella di salto	Paragrafo 6.2
Ricerca tabellare	
Numero di byte negativi	Paragrafo 9.1
Indice in tabella di lunghezza fissa	Paragrafo 9.3
Ordinamento a bolla	Paragrafo 9.3
Ricerca binaria in una tabella di lunghezza variabile	Paragrafo 9.3
Calcolo dell'indirizzo di un array	Paragrafo 9.3
Lista concatenata	Paragrafo 9.3

Operazioni del sistema: I/O, interruzioni e trappole

Chiamate di sistema di I/O	Paragrafo 7.6
Generazione di bit di parità	Paragrafo 8.3
Allocazione della memoria (TAS)	Paragrafo 8.3
Esempio d'inizializzazione del sistema	Paragrafo 10.3
Istruzione TRAP (chiamata a executive)	Paragrafo 11.2
Trappola di divisione per zero	Paragrafo 11.2
Trappola CHK e CHK2	Paragrafo 11.2
Trasferimento seriale di dati (MC68901)	Paragrafo 13.2
MOVES di supervisore a spazio di utente	Paragrafo 13.4

MATERIALE SUPPLEMENTARE

Relativamente a questo libro, i manuali del processore offrono una discussione più completa di certe caratteristiche del processore. I manuali per l'utente dell'MC68020, dell'MC68851 e dell'MC68881 sono disponibili dalla Prentice-Hall. I manuali per l'MC68030 e per altri chip possono essere ottenuti dalla Motorola, Inc.

Poiché i sistemi di sviluppo variano considerevolmente, il lettore dovrebbe fare riferimento ai manuali di consultazione per il sistema specifico su cui sono sviluppati i programmi. I suddetti manuali comprendono quello della programmazione in linguaggio assembler dello specifico assemblatore che si utilizza, come pure i manuali che descrivono l'impiego del sistema in operazioni quali l'editing, l'assemblaggio e l'esecuzione di programmi. Similmente, si dovrebbero utilizzare i manuali dell'hardware per il sistema e le specifiche di dati dei produttori per i singoli componenti, poiché tali documenti trattano i dettagli peculiari di qualsiasi elemento specifico.

RICONOSCIMENTI

Un certo numero di persone ha apportato contributi particolari a questo libro, il cui materiale è stato sviluppato in vari semestri e si è arricchito di molti utili suggerimenti da parte degli studenti del corso di Microcomputer Design alla University of Houston-Clear Lake. Anche lo staff del centro di calcolo dell'università si è mostrato molto ben disposto a collaborare per la stesura dei programmi di computer riportati in alcuni esempi del libro. Desidero ringraziare Dean E.T. Dickerson per il supporto finanziario e di altro tipo durante il lavoro. John Bahm ha fornito il suo contributo dedicando parte del suo tempo all'impegno d'installare nel suo laboratorio il computer basato sull'MC68020 utilizzato dall'autore.

Desidero ringraziare Ben LeDonne e Fritz Wilson della Motorola per la loro gradita assistenza. Patrick Adams della Quelo, Inc., ha donato i cross-assemblatori all'università. Il programma simulatore dell'MC68020 è stato fornito da Shelby Whatley della Big Bang Software, Inc. Ringrazio queste persone e le altre il cui supporto è stato prezioso.

La mia sincera riconoscenza va a tutti coloro che sono stati coinvolti, in un modo o nell'altro, nella produzione del libro. Ann Tran e Sheila Hemenway-Cropp hanno battuto a macchina il manoscritto. Faith Bryan ha proveduto ad una scrupolosa correzione delle bozze. Naturalmente, il personale della Prentice-Hall ha svolto il compito di porre il libro in produzione. È stato un vero piacere collaborare con Alice Barr ed i suoi assistenti.

Infine, vari amici e conoscenti devono essere ringraziati per la loro pazienza ed assistenza durante il lungo processo di produzione di un libro di testo. L'incoraggiamento di Karen Edwards è stato particolarmente apprezzato. Mi scuso con tutte le altre persone che hanno contribuito col loro impegno ma che non sono state citate qui. Sarà gradito qualsiasi commento o critica all'opera dell'autore, che potrà essere inviata alla University of Houston-Clear Lake, 2700 Bay Area Boulevard, Houston, Texas 77058.

T. L. Harman

CAPITOLO 1

INTRODUZIONE AI MICROPROCESSORI DELLA MOTOROLA

1.0 INTRODUZIONE

L'introduzione da parte della Motorola dei microprocessori MC68020 e MC68030 rappresenta un'evoluzione nelle famiglie dei moderni microprocessori di tale casa produttrice. I principali membri della famiglia sono l'MC68000 a 16 bit a l'MC68020 a 32 bit. Questi nuovi microprocessori a 32 bit estendono la capacità dei processori della Motorola di gestire applicazioni avanzate nella scienza e nella tecnica, come pure nel campo dell'informatica.

Un decennio di progressi nella tecnologia dei circuiti integrati fu coronato nel 1979 dall'annuncio della disponibilità commerciale del *microprocessore* MC68000 della Motorola. Questo microprocessore — e vari altri, che sono identificati complessivamente come *microprocessori a 16 bit* — è impiegato come unità centrale di elaborazione (*Central Processing Unit:* CPU) in alcuni tipi di computer, ma le sue dimensioni sono state ridotte al punto che le decine di migliaia di elementi circuitali che lo compongono sono contenute su un quadratino di silicio (*chip*) il cui lato è circa un quarto di pollice (6.4 millimetri). Dopo essere stato opportunamente incapsulato e connesso elettricamente ad altri circuiti integrati, che servono a memorizzare i dati e i programmi e comprendono la circuiteria d'ingresso/uscita, il chip del processore costituisce l'elemento centrale di un sistema di *microcomputer*. Le dimensioni del sistema possono essere piccole come 12 x 12 x 0.75 pollici (30 x 30 x 1.9 centimetri), prima di essere racchiuso in un contenitore per la protezione fisica. L'aggiunta di un alimentatore e degli opportuni dispositivi periferici, come un terminale con tubo a raggi catodici (*Cathode Ray Tube:* CRT) fornisce un sistema di elaborazione — un computer — in grado di essere programmato per svolgere una vasta gamma di compiti. Il microprocessore stesso controlla i dettagli di funzionamento del sistema a livello hardware ed esegue le operazioni aritmetiche, logiche o di altro tipo richieste dalla particolare applicazione. La Fig. 1.1 illustra gli elementi hardware fondamentali di un tipico sistema di microcomputer, utilizzabile per applicazioni personali o di piccole aziende.

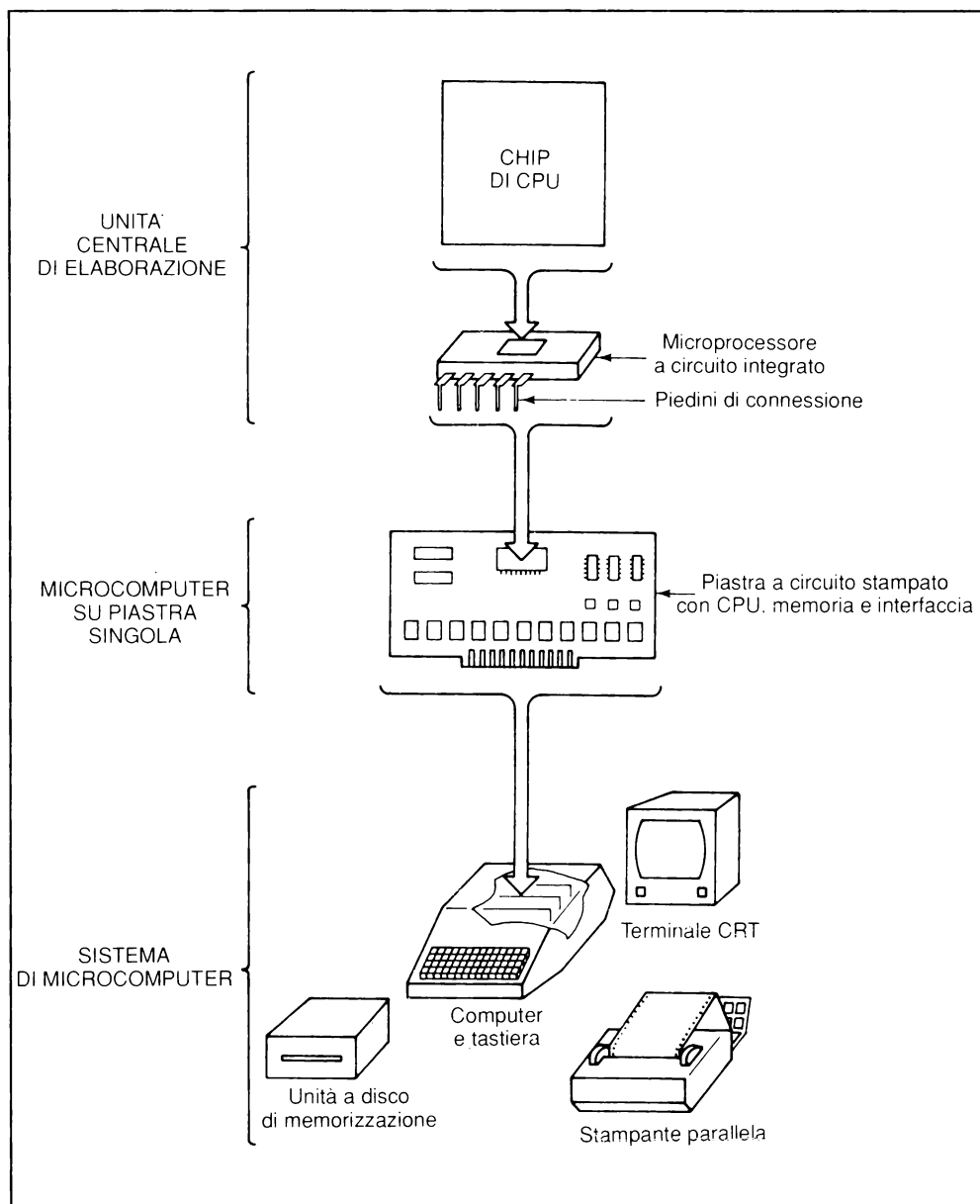



Fig. 1.1 *Un tipico sistema di microcomputer. (Per gentile concessione di Motorola, Inc.)*

L'entusiasmo generato dall'introduzione dell'MC68000 e di altri microprocessori della sua classe non era dovuto alla promessa di sistemi di elaborazione piccoli e poco costosi. Sistemi siffatti, basati su precedenti microprocessori, erano già disponibili in commercio. Piuttosto, la capacità notevomente potenziata di questi

microprocessori a 16 bit incanalò l'attenzione verso l'ampia varietà delle potenziali applicazioni dei microcomputer. Le capacità di questi processori sono paragonabili, e talvolta superiori, a quelle delle unità di elaborazione di molti minicomputer. La velocità di funzionamento — una caratteristica sussidiaria, ma comunque importante, di questi processori integrati — è confrontabile con quella di precedenti sistemi il cui costo era molte volte superiore. I progressi compiuti in alcuni decenni nel progetto dei processori e nella tecnologia d'integrazione dei circuiti hanno formato la base per la produzione di microprocessori in grado di gestire applicazioni con una vasta gamma di complessità e requisiti di velocità. Tali requisiti potevano essere soddisfatti in passato soltanto dai minicomputer.

Anche lo sviluppo di programmi (*software*) per questi microcomputer sta procedendo ad un ritmo sostenuto. È disponibile un certo numero di sistemi operativi e di programmi per computer basati sull'MC68000, che comprendono sia il software fornito dalla Motorola che quello creato da altri fornitori indipendenti. La combinazione del software con l'estesa linea di componenti hardware offerti dalla Motorola e da altri produttori formano una "famiglia" di prodotti che forniscono un supporto allo sviluppo di sistemi basati sull'MC68000.

I microprocessori a 32 bit MC68020 e MC68030. Quando fu presentato l'MC68020 nel 1984, l'MC68000 a 16 bit si era già affermato come microprocessore. L'MC68020 estendeva le caratteristiche di base dell'MC68000, costituendo un microprocessore potenziato nella classe dei 32 bit, in quanto era dotato di molte capacità non possedute dai precedenti processori a 16 bit. Un importante miglioramento era la capacità dell'MC68020 di essere combinato con coprocessori quali il coprocessore matematico in virgola mobile (MC68881) ed il coprocessore di gestione della memoria (MC68851). Come mostrato nella Fig. 1.2, l'insieme dei tre chip, incorporato in un modulo di computer su una sola piastra, può costituire un'unità di elaborazione completa su una singola piastra a circuito stampato. Questo modulo può fungere da processore centrale e da unità di controllo di un sofisticato sistema di elaborazione.

L'MC68030 si spinge di un passo avanti rispetto all'MC68020, combinando su un singolo chip una versione migliorata dell'MC68020 con la capacità di gestione della memoria dell'MC68851. Tale combinazione accresce le prestazioni di un sistema con l'ulteriore vantaggio di un numero ridotto di componenti. Inoltre, l'MC68030 può essere affiancato da un coprocessore matematico MC68882, per costituire un insieme di due chip in grado di operare come un'unità di elaborazione completa.

In questo capitolo introduttivo, sarà presentato un certo numero di computer che utilizzano l'MC68020 come unità centrale di elaborazione. Questi prodotti includono personal computer, stazioni di lavoro (*workstation*) d'ingegneria, computer multiprocessore e potenti sistemi di elaborazione in grado di provvedere allo sviluppo software di programmi per qualsiasi applicazione. In questo capitolo, sarà descritto anche il modo in cui lo sviluppo di tali prodotti è notevolmente facilitato dal supporto hardware e software per le famiglie di microprocessori della Motorola.

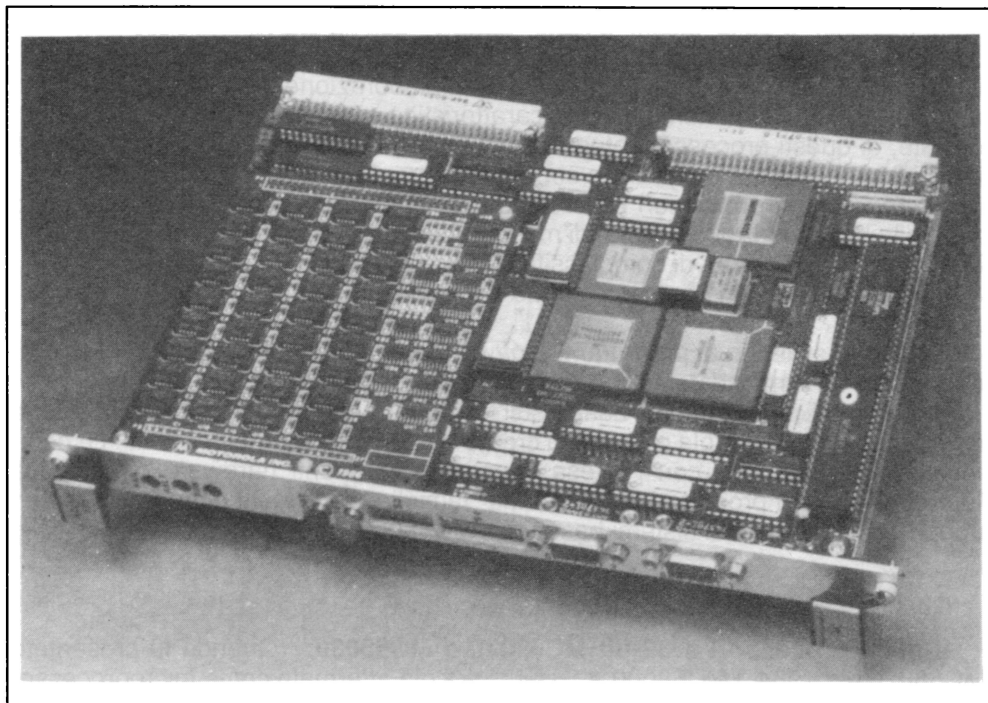



Fig. 1.2 *MC68020/MC68851/MC68881: un insieme di tre chip su un computer a piastra singola MVME135 della Motorola. (Per gentile concessione di Motorola, Inc.)*

1.1 APPLICAZIONI DELL'MC68020 DELLA MOTOROLA

Nonostante le sue piccole dimensioni fisiche, il processore MC68020 della Motorola dispone delle capacità e della velocità operativa necessarie per funzionare come processore centrale in un computer progettato per applicazioni molto avanzate. In questo paragrafo viene presentato un certo numero di esempi di tali applicazioni, per dare un'idea della vasta gamma di prodotti che si possono progettare basandosi sull'MC68020. Gli esempi descritti sono elencati nella Tab. 1.1, che indica l'area applicativa, l'applicazione specifica e il produttore. Nella maggioranza dei casi, l'esempio menzionato rappresenta una crescita della capacità del prodotto ad un costo inferiore rispetto a quello di prodotti simili progettati e realizzati usando la tecnologia di computer disponibile prima dell'introduzione dell'MC68020.

In effetti, alcuni produttori elencati nella Tab. 1.1 hanno già offerto in passato prodotti simili, basati sul precedente MC68000 a 16 bit. Dopo l'introduzione dell'MC68020 da parte della Motorola, i progettisti hanno incorporato l'MC68020 nei nuovi progetti, al fine di migliorare le prestazioni dei prodotti. Nei prossimi capitoli saranno discusse molte delle capacità dell'MC68020 che hanno consentito ai produttori di sviluppare tali prodotti.

Tab. 1.1 *Applicazioni dell'MC68020.*

AREA	APPLICAZIONE	PRODUTTORE
Istruzione	Ausilio all'insegnamento o modulo di valutazione	Motorola Semiconductor Products, Phoenix, Ariz.
Personal computer	Programmazione	Apple Computers, Cupertino, Calif.
Workstation	Ingegneria	Apollo Computers, Chelmsford, Mass.
Multiprocessore	Scienza e ingegneria	Masscomp, Westford, Mass.
Sistema di sviluppo	Sviluppo di sofware e di hardware	Motorola Semiconductor Phoenix, Ariz.

1.1.1 Istruzione

La Fig. 1.3 mostra una fotografia della piastra di processore MVME133 della Motorola. Con l'aggiunta di un alimentatore e di un terminale per l'operatore, questo modulo può essere usato come un computer su una sola piastra. Tale piastra ha una CPU MC68020, un coprocessore MC68881 e 1 MB (1048576 byte) di memoria. Nella fotografia, l'MC68020 è il più grande contenitore di circuito integrato presente sulla piastra. Esso è collegato agli altri elementi del sistema mediante piste di rame per la segnalazione ed il trasferimento dei dati. Nell'insieme, queste linee di segnale formano il bus interno per il modulo su piastra singola. Altri dispositivi a circuiti integrati sulla piastra sono costituiti dai chip di memoria ad accesso casuale (*Random Access Memory:* RAM) e dai componenti d'interfaccia. L'utente comunica col modulo per mezzo di un terminale collegato tramite cavo ad un connettore frontale della piastra.

I grandi connettori posteriori della piastra sono quelli di VMEbus. Essi consentono la connessione della piastra con altre piastre VME-compatibili, per formare una parte di un sistema più grande. In quest'applicazione, l'MVME133 diviene la piastra di controllore per l'intero sistema del computer. Tali sistemi saranno discussi nel par. 1.2.

Come computer su singola piastra (*"single-board"*), la piastra dell'MVME133 può essere impiegata per valutare le caratteristiche dell'MC68020 e dell'MC68881 durante lo sviluppo e l'esecuzione dei programmi. Questi programmi possono essere creati usando un sistema operativo elementare, denominato *monitor*, registrato nei chip di memoria a sola lettura (*Read-Only Memory:* ROM) presenti nel modulo. È disponibile un'assemblatore di una sola riga per consentire all'utente di

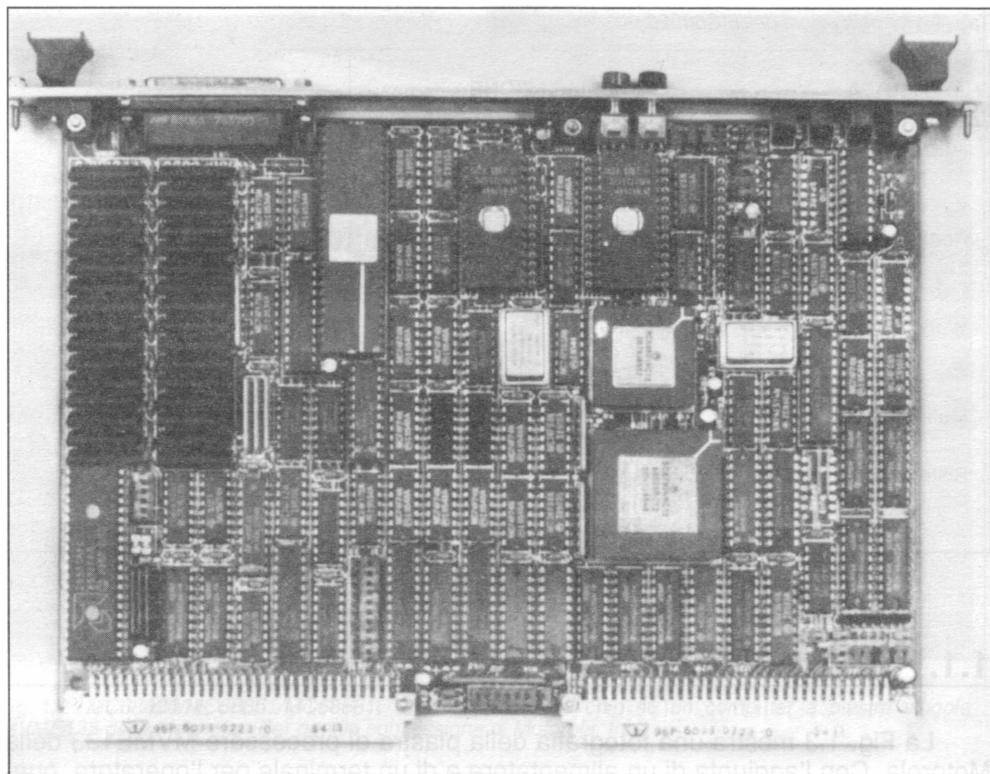



Fig. 1.3 *La piastra del processore MVME133. (Per gentile concessione di Motorola, Inc.)*

creare semplici programmi. In alternativa, i programmi possono essere creati su un altro sistema di computer, che sia in grado di generare del codice in linguaggio-macchina per l'MC68020. Il codice eseguibile potrà quindi essere caricato nella memoria del modulo MVME133 per essere eseguito. Entrambi questi metodi per lo sviluppo del software sono descritti altrove nel libro. Per il momento, si dovrebbe notare che l'MVME133 rappresenta l'hardware necessario per fornire la CPU, il coprocessore, la memoria e le funzioni d'interfacciamento di un sistema di computer su una singola piastra a circuito stampato.

1.1.2 Personal computer

Per quanto le definizioni possano variare, il personal computer è generalmente considerato un sistema di elaborazione completo, con la flessibilità di eseguire un'estesa varietà di programmi per applicazioni tecniche, aziendali e domestiche. Inoltre, il suo prezzo ridotto lo rende accessibile al singolo acquirente. Grazie ai progressi compiuti nella tecnologia dei computer, i "personal" si sono evoluti dai sistemi relativamente semplici, che impiegavano un'unità centrale di elaborazione a

8 bit, fino ai potenti computer che utilizzano CPU a 32 bit come l'MC68020. Il modello Macintosh II della Apple Computer, illustrato nella Fig. 1.4, è un personal computer che impiega l'MC68020 ed il coprocessore in virgola mobile MC68881 per l'elaborazione. Questo computer ha estese capacità di visualizzazione grafica, basate sul gran numero di programmi disponibili per applicazioni specifiche. Per esempio, vari programmi offerti dalla Apple e da altre società consentono di utilizzare il sistema per la stesura di testi e la preparazione di documenti, per lo sviluppo di software, o come una elementare workstation d'ingegneria. I precedenti prodotti della Apple Computer Inc., come la macchina Lisa ed il computer Macintosh originale, avevano capacità simili, ma impiegavano come CPU l'MC68000 a 16 bit.

A causa della compatibilità verso l'alto dell'MC68000 con l'MC68020, i programmi scritti per i computer Apple che incorporano l'MC68000 possono essere eseguiti sul Macintosh II, basato sull'MC68020, senza grandi modifiche. Questa è una caratteristica importante per coloro che possiedono una grande raccolta di programmi di precedenti computer Apple basati sull'MC68000 come il Macintosh. Quando vengono trasportati sul Macintosh II, in genere tali programmi saranno eseguiti molto più velocemente; normalmente si assiste ad una riduzione di un fattore 3 o 4 del tempo di esecuzione. Quando la compatibilità non è richiesta, i programmi per nuove applicazioni possono sfruttare appieno la potenza di elaborazione della CPU a 32 bit.

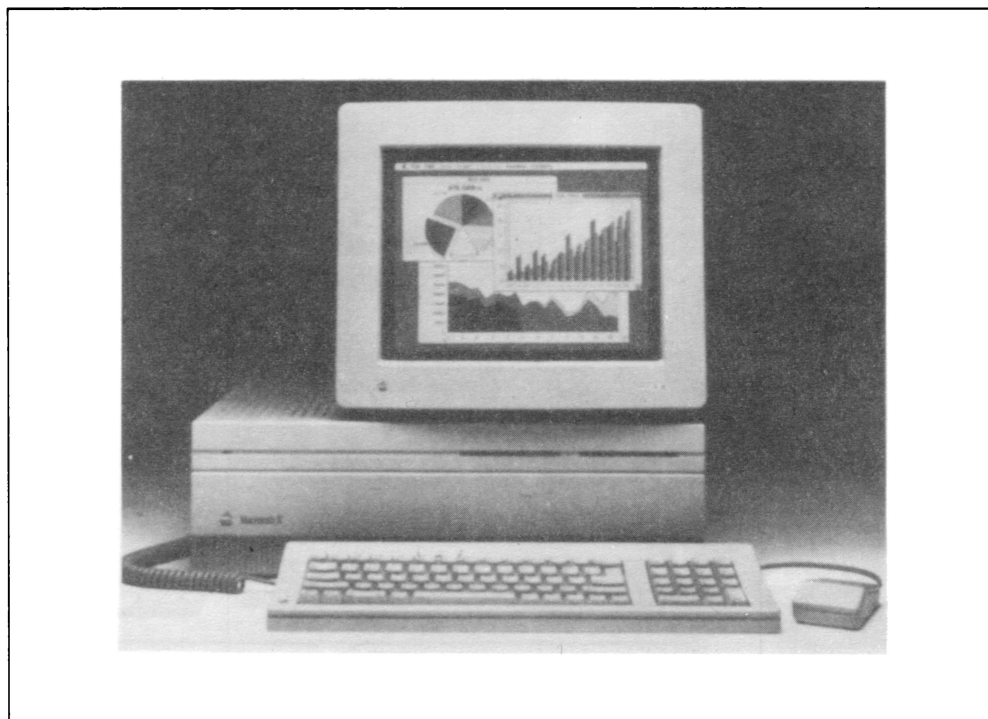



Fig. 1.4 *Apple Macintosh II. (Per gentile concessione di Apple Computer, Inc.)*

1.1.3 Workstation d'ingegneria

La *workstation* (stazione di lavoro) è un'applicazione importante dei microprocessori a 32 bit. I prodotti disponibili variano dai computer da scrivania poco costosi per applicazioni commerciali e gestionali, fino alle potenti workstation per la soluzione di problemi scientifici di analisi numerica o di ingegneria assistita dal computer (*Computer Aided Engineering:* CAE). In genere, le stazioni di lavoro d'ingegneria hanno estese capacità di visualizzazione grafica e consentono il collegamento in rete. In una rete, una workstation può operare con un certo numero di altri computer, condividendo sia le informazioni che i dispositivi periferici come le grandi unità di memorizzazione su disco.

Un certo numero di produttori offre workstation d'ingegneria che impiegano l'MC68020 come processore centrale. Queste workstation possono essere usate per varie applicazioni, tra cui il progetto di circuiti integrati, lo studio scientifico di modelli con l'impiego di grafica 3D (tridimensionale), e l'analisi strutturale. Il sistema Apollo mostrato in Fig. 1.5 rappresenta un esempio di tale classe di workstation. La nuova linea di prodotti di workstation Apollo si è evoluta da prodotti precedenti basati sull'MC68000.

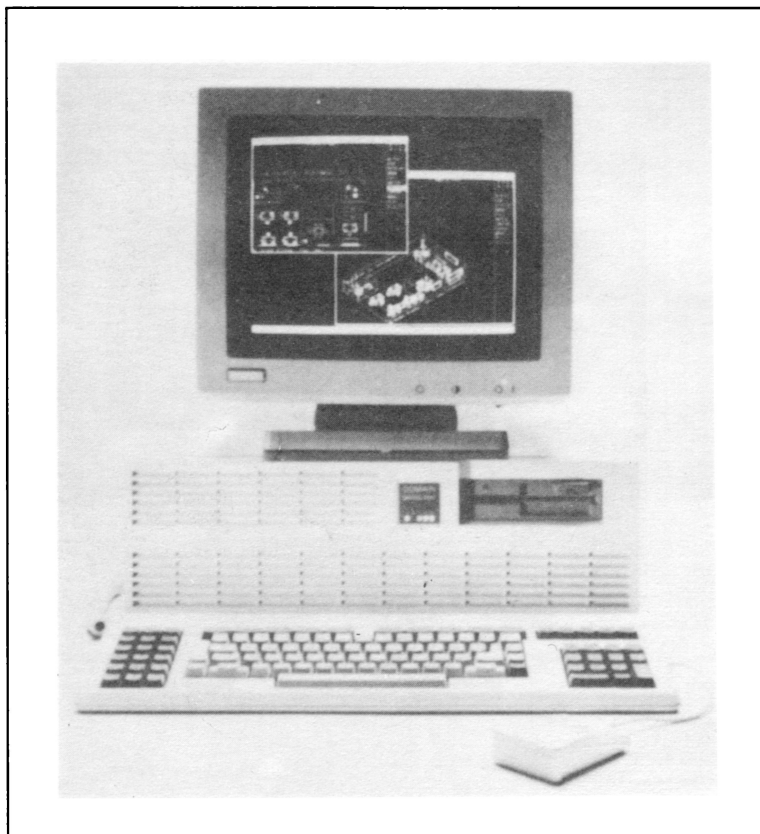



Fig. 1.5
Workstation Apollo. (Per gentile concessione di Apollo Computer, Inc.)

1.1.4 Sistemi multiprocessore

Una nuova generazione di computer, talvolta denotati "super-microcomputer", è stata progettata basandosi sui microprocessori a 32 bit come l'MC68020. Questi computer sono utilizzati sia per il calcolo scientifico e tecnico che per l'elaborazione delle informazioni. La famiglia di computer Masscomp, mostrata in Fig. 1.6, è composta da workstation con un singolo processore, come pure da computer multiprocessore. Il modello più grande trova impiego in applicazioni in cui sono necessari più processori in un unico sistema per soddisfare i requisiti di velocità e di potenza di calcolo di un problema scientifico.

In un'applicazione tipica, il computer multiprocessore può essere istruito per eseguire più programmi simultaneamente, ciascuno su un distinto processore del sistema. Per risolvere altri problemi, si può suddividere un singolo programma in modo che le sue parti separate possano essere eseguite da differenti processori. Entrambi questi metodi riducono il tempo richiesto per eseguire i programmi, rispetto al tempo impiegato dal computer con un singolo processore.

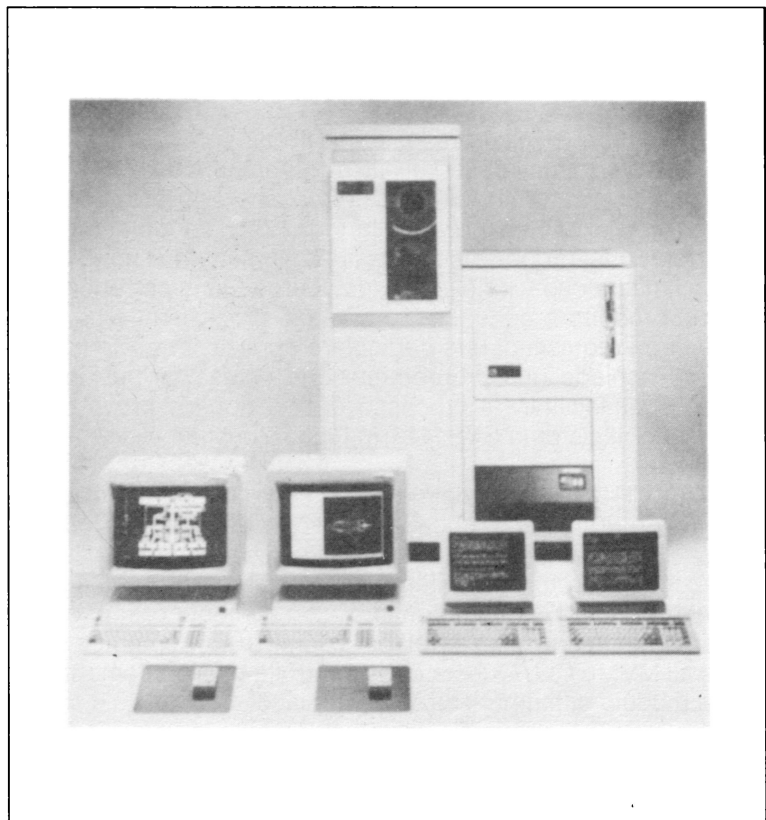



Fig. 1.6
La famiglia di computer Masscomp. (Peg gentile concessione di Masscomp.)

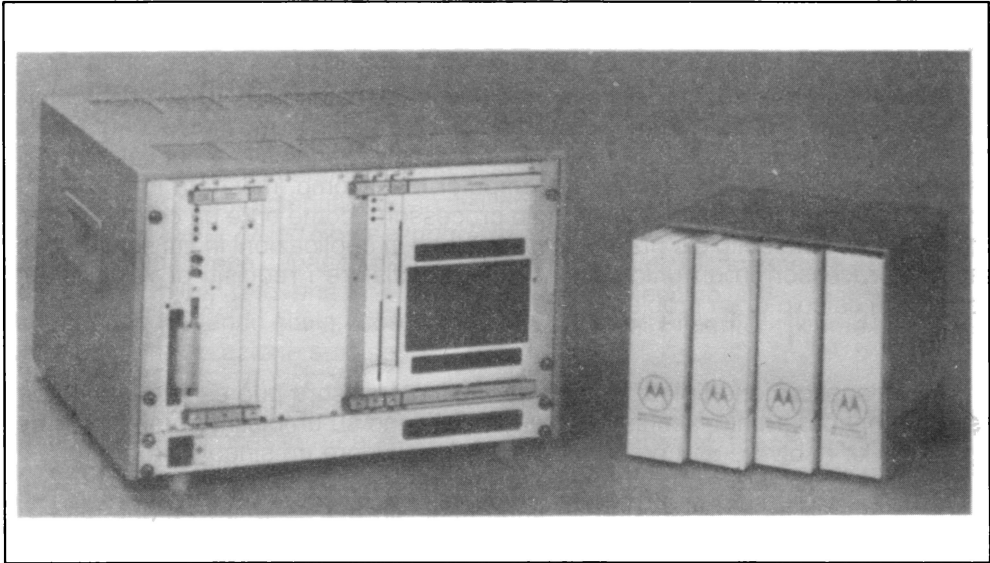



Fig. 1.7 *Il sistema di sviluppo SYS1131 della Motorola. (Per gentile concessione di Motorola, Inc.)*

1.1.5 Sistemi di sviluppo

I produttori di microprocessori a 32 bit vendono normalmente altre apparecchiature e programmi destinati allo sviluppo hardware e software di prodotti o computer che utilizzano come CPU il processore a 32 bit.[1] Per esempio, il modulo MVME133 di CPU, mostrato nella Fig. 1.3, è venduto come computer su singola piastra per quei progettisti che desiderano soltanto l'unità centrale di elaborazione ed una limitata capacità di memoria. Il progettista potrebbe incorporare tale modulo in un prodotto a cui vengono aggiunti programmi e hardware specializzati per soddisfare i requisiti di una particolare applicazione. Se il progetto di sviluppo del software richiede una programmazione estesa, si può impiegare un *sistema di sviluppo* per facilitare il processo di creazione e prova di qualsiasi programma necessario. Il sistema SYS1131 in Fig. 1.7 è stato realizzato a tal fine.

Il SYS1131 è essenzialmente un computer adattato allo sviluppo di software. Per quei programmatori che creano programmi per l'MC68020, il sistema fornisce la capacità di memoria ed il software necessari per tale sviluppo. Con l'aggiunta dei terminali di operatore, il sistema può gestire otto utenti simultaneamente. Fisicamente il SYS1131 è un contenitore (con alimentatore) in cui sono inseriti vari moduli di computer. L'unità di elaborazione è un computer su una singola piastra, simile all'MVME133. Questa unità controlla gli altri elementi del sistema, ad esempio un modulo di memorizzazione su disco.

[1] Un certo numero di società (oltre alla Motorola) vende prodotti di supporto ai progettisti di hardware e di software i quali utilizzano l'MC68020.

Un sistema operativo ed altri programmi di supporto allo sviluppo del software sono memorizzati sull'unità a disco. Questi programmi vengono trasferiti alla memoria ed eseguiti in conformità coi comandi dell'operatore.

1.2 I MICROPROCESSORI DELLA MOTOROLA ED IL CONCETTO DI "FAMIGLIA"

L'MC68020 fu scelto come unità centrale di elaborazione nei prodotti descritti nel par. 1.1 per varie considerazioni economiche e tecniche. Le capacità specifiche dell'MC68020 erano senza dubbio importanti nella scelta, specialmente quando il prodotto finito presentava delle prestazioni migliorate rispetto ai predecessori simili che impiegavano un processore meno potente. Per la maggior parte dei produttori, un altro fattore di importanza capitale nella scelta è il *supporto* fornito alla linea di processori. Questo supporto è reso disponibile dal produttore del processore e da altri fornitori. Tale supporto consiste di hardware, software, sistemi di sviluppo, e di altre componenti; esso viene fornito per consentire alle aziende produttrici di progettare, realizzare, collaudare e infine produrre i loro prodotti nella maniera più economica. Inoltre, via via che i progressi tecnologici consentono di migliorare le prestazioni e di ridurre i costi del prodotto, quest'ultimo dev'essere modificato dal produttore in vari modi per poter restare competitivo.

Il concetto di *famiglia*, per quanto concerne i sistemi di microcomputer, garantisce che la linea di processori sia dotata di un supporto adeguato e che venga progressivamente migliorata. Chi conosce il processore di base (MC68000) non avrà molte difficoltà ad apprendere le caratteristiche dei processori più recenti e le varie possibilità offerte dagli altri membri della famiglia.

La famiglia di microprocessori a 16 bit della Motorola comprende l'MC68000, l'MC68010 e l'MC68012. I processori a 32 bit sono l'MC68020 e l'MC68030. Dopo aver discusso la linea completa di microprocessori "imparentati" con l'MC68000, questo libro tratterà in primo luogo la famiglia di processori a 32 bit.[2] Questi processori hanno 32 linee di segnale per trasferire i dati e 32 linee di segnale per indirizzare la memoria.

La Tab. 1.2 è un riepilogo di molti dei criteri di supporto discussi in questo paragrafo. La linea di circuiti integrati di processore comprende i processori, i circuiti per facilitare la progettazione di interfacce e i dispositivi speciali per potenziare le prestazioni di un sistema di computer. Il supporto software per la famiglia di microprocessori include i sistemi operativi, i programmi di sviluppo ed altri programmi

[2] Nella letteratura tecnica della Motorola, l'intera linea di microprocessori a 16 bit e a 32 bit e dei chip periferici è talvolta designata come la famiglia "MC68000". Le parti a 16 bit comprendono la famiglia dell'MC68000, mentre i componenti a 32 bit sono denotati come la famiglia dell'MC68020.

Tab. 1.2 *Supporto per la famiglia di microprocessori della Motorola e di altre società.*

Tipo	Supporto
PROCESSORI E CHIP DI SUPPORTO	
Microprocessori a 16 bit e a 32 bit	Processore di base e versioni di base disponibili in vari contenitori e differenti velocità operative
Circuiti d'interfaccia periferica	Circuiti per l'interfacciamento della CPU ad una vasta gamma di dispositivi periferici
Dispositivi speciali e coprocessori	Dispositivi per la matematica in virgola mobile, controllo di rete, ed altre applicazioni
SOFTWARE	
Sistemi operativi	Vari sistemi operativi per applicazioni in tempo reale, time-sharing o speciali
Software di sviluppo	Programmi di editing, assemblaggio, compilazione e debugging
Software applicativo	Programmi speciali per contabilità, analisi d'ingegneria, ecc.
Documentazione	Manuali, note applicative, specifiche
SVILUPPO	
Sistemi di sviluppo	Sistemi completi per lo sviluppo di software e per l'integrazione hardware/software
Moduli di computer su piastra singola	Unità di elaborazione, moduli di memoria, ed altri sottosistemi hardware completi

denotati collettivamente come *software di sistema*, come pure certi programmi applicativi. Per quegli utenti che sviluppano software applicativo, è disponibile un certo numero di sistemi di sviluppo per facilitare la produzione del software. Oltre a consentire la creazione e il collaudo del software, alcuni sistemi di sviluppo servono nell'integrazione del software di applicazioni col prototipo dell'hardware sviluppato dall'utente. Questa capacità è d'importanza capitale se il prodotto finito è un sistema completo, com'era il caso di molti esempi presentati nel par. 1.1.

1.2.1 Il processore MC68020 e relativi chip

Un'unità di elaborazione di un microcomputer consiste della CPU e di un certo numero di circuiti integrati (*chip*) ausiliari per l'interfacciamento, ma non include la memoria. I chip sono interconnessi su una piastra circuitale che somiglia di solito al modulo MVME133 della Motorola illustrato nel par. 1.1. La capacità globale e la velocità operativa del sistema sono essenzialmente determinate dalle caratteristiche della CPU, quando non sono richiesti trasferimenti d'ingresso/uscita (*Input/Output:* I/O) dei dati durante l'esecuzione di un programma. Quando invece i trasferimenti di I/O sono necessari, la flessibilità del sistema di comunicare coi vari dispositivi periferici dipende dal tipo di chip d'interfaccia di cui il sistema dispone. Per consentire la massima varietà di applicazioni, la Motorola ed altri fornitori producono un certo numero di distinte unità centrali di elaborazione e di chip d'interfaccia. Man mano che la famiglia di chip crescerà, diverranno disponibili processori potenziati ed ulteriori chip d'interfaccia.

La famiglia a 16 bit. I produttori di processori rispondono alle esigenze crescenti delle aziende di prodotti finiti ed ai progressi tecnologici arricchendo il progetto di un microprocessore come l'MC68000 e fornendo nuove versioni. Le versioni successive, come l'MC68010, si distinguono da quella di base (MC68000) per la rispettiva designazione numerica. Normalmente questi processori sono compatibili con l'MC68000 per molti aspetti, tuttavia presentano caratteristiche diverse. Per esempio, i membri della famiglia del processore MC68000 sono disponibili con differenti contenitori fisici, differenti intervalli di temperatura operativa e con varie velocità operative. Per il resto, i rispettivi insiemi di istruzioni e le caratteristiche elettriche sono identiche a quelle dell'MC68000. Altri processori della famiglia possono mostrare differenze più significative, come nel caso di parecchie versioni avanzate dell'MC68000. L'evoluzione della famiglia in funzione dell'anno di presentazione di ciascun processore è mostrata in Fig. 1.8. La Tab. 1.3 descrive le caratteristiche delle varie versioni.

L'intervallo di velocità operative disponibile nella famiglia MC68000 è evidente nei processori Motorola designati come MC68000L4, MC68000L6, LC68000L8, MC68000L10 e MC68000L12. L'ultima designazione numerica indica (in milioni) il massimo numero di cicli di clock hardware per secondo (L12 rappresenta 12 milioni di cicli di clock al secondo). Per esempio, il dispositivo L12 eseguirà il medesimo programma ad una velocità 1.25 (12.5/10) volte superiore a quella dell'L10, o 1.56 (12.5/8) volte più velocemente dell'L8, e così via. In un dato prodotto, la sostituzione della CPU con un processore più veloce (o più lento) modificherà in modo proporzionale le prestazioni, se la velocità operativa è determinata soltanto dal processore.

La Motorola ha anche introdotto l'MC68008, che è una versione con bus dati di 8 bit dell'MC68000. Questo processore conserva la maggior parte delle caratteristiche dell'MC68000 originale, ma è progettato per applicazioni che impiegano trasferimenti di dati di 8 bit. Questa riduzione del numero di linee di segnale di dati riduce il costo di un sistema o di un prodotto e semplifica l'interfaccia con certe

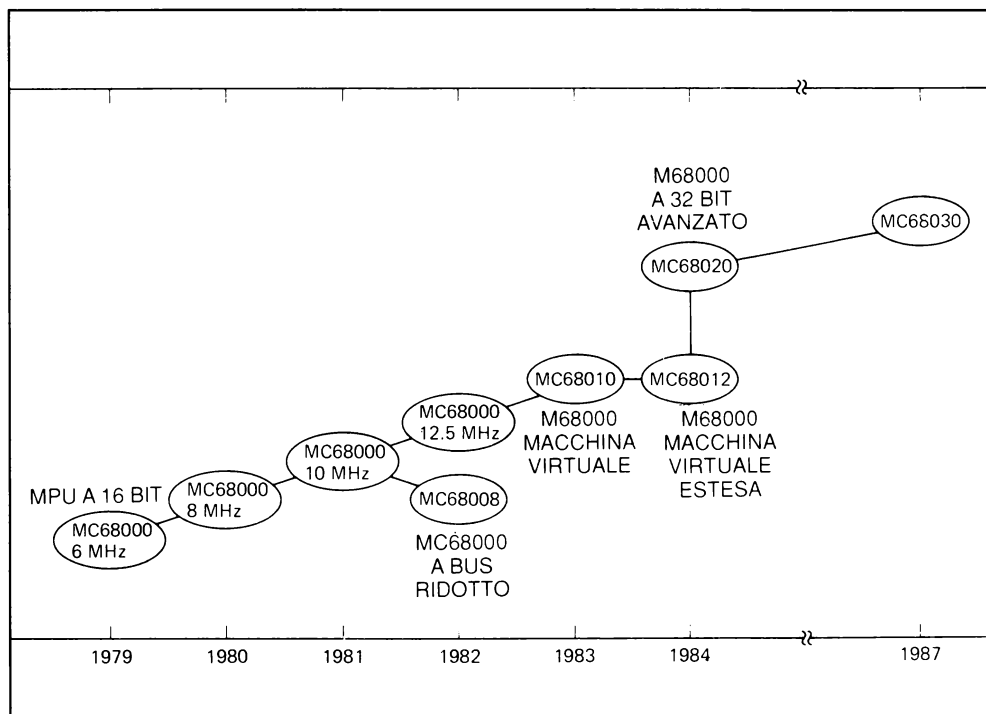



Fig. 1.8 *Evoluzione dei microprocessori della Motorola.*

Tab. 1.3 *Processori e coprocessori della Motorola.*

Processore	**Caratteristiche d'impiego**
Famiglia a 16 bit	
MC68000L4, MC68000L6 MC68000L8, MC68000L10 MC68000L12	Differenti velocità operative, indicate dal suffisso, in milioni di cicli di clock al secondo
MC68008	Bus di dati di 8 bit
MC68010 MC68012	Capacità di gestire la memoria virtuale
Famiglia a 32 bit	
MC68020	Capacità di trasferire e indirizzare dati di 32 bit
MC68030	Gestione della memoria su chip
MC68851	Coprocessore di gestione della memoria
MC68881/MC68882	Coprocessori di aritmetica in virgola mobile

unità periferiche. Un vantaggio importante sui precedenti processori a 8 bit in un'applicazione di questo tipo è dato dal fatto che l'MC68008 esegue i programmi scritti per l'MC68000.

L'MC68010 è una versione potenziata dell'MC68000, progettata essenzialmente per i sistemi con memoria virtuale. In quest'applicazione, la dimensione di un programma non è limitata dalla capacità della memoria fisica, poiché una parte del programma può essere memorizzata in un dispositivo esterno quale un'unità a disco. Le porzioni del programma memorizzate sul disco vengono trasferite alla memoria principale quando se ne presenta la necessità. Quindi il processore è in grado di proseguire l'esecuzione delle istruzioni dopo che il nuovo segmento di programma è stato trasferito nella memoria principale. L'MC68010 e l'MC68020 hanno la medesima capacità fondamentale per i sistemi di memoria virtuale, come sarà descritto nel cap. 12. Come l'MC68000, anche l'MC68010 è disponibile con varie velocità operative.[3]

La famiglia a 32 bit. La famiglia di processori e coprocessori a 32 bit della Motorola comprende l'insieme di chip MC68020/MC68881/MC68851 ed i chip MC68030/MC68882. L'MC68020 è disponibile con velocità operative di 12.5, 16.67, 20, 25 e 33.3 milioni di operazioni (cicli di clock) al secondo. I due coprocessori eseguono l'aritmetica in virgola mobile e la gestione della memoria, rispettivamente. Un MC68020 è compatibile con l'MC68000 per molti aspetti ma è dotato di varie migliorie per poter gestire più efficacemente i moderni sistemi operativi e il software o l'hardware per applicazioni speciali.

Dal punto di vista dei programmatori, l'MC68030 ed il suo coprocessore MC68882 sono compatibili per molti aspetti con l'MC68020 ed i suoi coprocessori. L'MC68030 ha sul suo chip la propria unità di gestione della memoria, per accelerare le conversioni d'indirizzo richieste nei sistemi con memoria virtuale. Inoltre questo processore è stato progettato con un maggior grado di parallelismo rispetto all'MC68020. Per esempio, il funzionamento "concorrente" (simultaneo) della CPU e del coprocessore matematico MC68882 è possibile con l'MC68030. Il capitolo 15 è dedicato all'MC68030.

Dispositivi d'interfacciamento per la famiglia MC68020. Nella Tab. 1.4 sono elencati alcuni dei numerosi tipi di dispositivi d'interfacciamento che si presentano fisicamente come "chip" di circuiti integrati per la famiglia della Motorola. I chip d'ingresso/uscita sono dispositivi d'interfacciamento di tipo generale, che servono a connettere la CPU ad una grande varietà di unità periferiche. Questi chip sono programmabili per fornire flessibili capacità d'ingresso/uscita sotto il controllo dell'unità centrale di elaborazione. Anche gli altri dispositivi d'interfacciamento elencati nella Tab. 1.4 sono programmabili e ognuno di tali chip risponde ad istruzioni codificate secondo il suo progetto e la sua funzione.

[3] Altri processori nella famiglia a 16 bit sono l'MC68012 e l'MC68HC000. Una CPU MC68012 è un MC68010 con capacità d'indirizzamento accresciute (31 linee d'indirizzo anziché 24) ed altre caratteristiche speciali. La designazione HC per l'MC68000 indica l'impiego della tecnologia CMOS, che richiede circa un decimo della potenza consumata dall'MC68000 standard.

Tab. 1.4 *Dispositivi d'interfacciamento per l'MC68020.*

Applicazione	Esempi di chip d'interfacciamento
Ingresso/uscita (I/O) parallelo e seriale	Dispositivi d'interfaccia generici (MC68230, MC68901)
Controllore di DMA	Accesso diretto alla memoria (MC68450)
Comunicazioni e reti locali	Controllore e dispositivi di comunicazione (MC68605, MC68824, MC68184)
Controllo di sistemi	Controllo dell'interruzione e arbitrato di bus (MC68153, MC68452)

Nota: Ogni dispositivo d'interfacciamento elencato nella Tab. 1.4 è prodotto come un circuito integrato su un singolo chip. Nel gergo industriale, sono noti come "chip" d'interfacciamento.

La Motorola, come altre case produttrici, produce chip per molte applicazioni ordinarie, come quelle elencate nella Tab. 1.4. I chip sono progettati per essere compatibili col processore centrale, sia elettricamente che rispetto alle convenzioni di programmazione per la famiglia dell'MC68020. Purtroppo è difficile caratterizzare in maniera semplice tali chip d'interfacciamento, poiché ognuno di essi ha istruzioni speciali relative alla sua funzione particolare. Il loro scopo — come discusso qui — è quello di semplificare l'interfaccia tra la CPU ed altre unità di un sistema. In tal modo, quando è disposibile un chip adatto, il progettista dell'interfaccia viene sollevato dall'impegno di dover progettare e collaudare la relativa circuiteria d'interfacciamento.

1.2.2 Supporto di software per la famiglia dell'MC68020

Dopo che l'hardware di un sistema di computer è stato costruito, devono essere sviluppati dei programmi per controllare il funzionamento complessivo del sistema. I vari programmi che sono eseguiti dal microcomputer sono denotati genericamente come "software", per distiguerli dall'apparato fisico — "hardware" — del sistema. Per i fini di questo libro, è opportuno considerare tre categorie o "livelli" di software, oltre al livello dell'hardware, come mostrato nella Tab. 1.5.

Al livello più prossimo all'hardware, il *sistema operativo* gestisce le risorse hardware del sistema. Il sistema operativo è frequentemente denominato programma esecutivo (*executive*) o *supervisore.* Al livello successivo è situato il *software di*

Tab. 1.5 *Livelli di software.*

Livello	Esempi
Software applicativo	Programmi adattati per risolvere un problema specifico
Software di sviluppo	Editor, assemblatori o compilatori per creare programmi applicativi
Sistema operativo	Programma per controllare la CPU, l'ingresso/uscita e la memorizzazione su disco (file)
Livello hardware	CPU, chip d'interfacciamento, memoria e dispositivi periferici

sviluppo, utilizzato dal programmatore per creare e collaudare i programmi applicativi. Il *software applicativo* è scritto "su misura" per il sistema del computer, affinché questo sia in grado di svolgere un compito specifico. Nella famiglia dell'MC68020, un vasto elenco di prodotti software è disponibile sia dalla Motorola che da fornitori indipendenti di software. La discussione in questo sottoparagrafo è ristretta ai sistemi operativi e al software di sviluppo fornito dalla Motorola, come definito nella Tab. 1.6.

Sistemi operativi della Motorola. Il sistema operativo (*Operating System*) VERSAdos™* può essere fornito dalla Motorola col sistema di computer SYS1131 descritto nel par. 1.1. Si tratta di un sistema operativo multiutente, con estese capacità di gestione dei file, per la memorizzazione di file di programmi e di dati sulle unità a disco. È considerato un sistema operativo di tipo generale, poiché funge da supporto allo sviluppo dei programmi e può anche essere usato come sistema operativo per applicazioni che impiegano i computer della Motorola.

Per un'applicazione specifica, è disponibile il sistema operativo in tempo reale (*Real-Time Operating System*) RMS68K, che consente al computer di rispondere rapidamente ad eventi esterni, come richiesto da un'applicazione di controllo di processo. Lo scopo primario dell'RMS68K è quello di fornire un supporto all'esecuzione dei programmi applicativi richiesti in una situazione di tempo reale.

Il sistema operativo System V/68 è derivato da UNIX™*. Questo sistema operativo multiutente è molto diffuso tra i sistemi di sviluppo, particolarmente quelli che impiegano un compilatore di linguaggio C.

* RMS68K, System V68 e VERSAdos sono marchi registrati di Motorola, Inc.
* UNIX è un marchio registrato di AT&T Technologies.

Tab. 1.6 *Software di sistema disponibile dalla Motorola.*

Software di sistema	Impiego
SISTEMI OPÈRATIVI	
VERSAdos	Sistema operativo generale
RMS68K	Executive in tempo reale
System V/68	Sistema operativo derivato da UNIX
SOFTWARE DI SVILUPPO	
Editor di testo	Creazione e modifica di programmi
Assemblatore e programma di debugging	Traduzione e collaudo di programmi in linguaggio assembler
Compilatori di FORTRAN, di Pascal e di C	Compilazione di programmi in linguaggio ad alto livello
Editor di collegamento	Collegamento di moduli separati di programmi in linguaggio-macchina
CROSS-SOFTWARE	
Cross-assemblatori e cross-compilatori	Creazione su altro computer di programmi in linguaggio-macchina di MC68020

Software di sviluppo. Sotto il controllo del sistema operativo, vari programmi di ausilio allo sviluppo del software possono essere eseguiti su un sistema di computer di tipo generale. Il software offerto dalla Motorola per lo sviluppo, come elencato nella Tab. 1.6, comprende un editor di testo, alcuni traduttori di linguaggi e vari programmi di servizio ("utilities"). L'editor di testo è usato per la creazione e la modifica del programma in fase di sviluppo, prima che esso venga tradotto nelle espressioni del linguaggio-macchina dagli opportuni traduttori di linguaggio. Un assemblatore e vari compilatori di linguaggi ad alto livello possono essere acquistati dalla Motorola, che offre quindi un'ampia scelta di linguaggi di computer.

Diversi programmi speciali, spesso denotati come programmi di servizio o "utilities", costituiscono un supplemento al software di sviluppo per l'MC68020. Un programma di debugging è uno strumento utile per individuare gli errori di programmazione (i "bachi"), in quanto permette di eseguire una piccola porzione del programma e di esaminarne gli effetti. L'editor di collegamento (*linkage editor*) serve

a combinare più moduli di programma che sono stati assemblati (o compilati) separatamente, in modo da creare un programma completo, pronto per essere eseguito dall'MC68020.

Durante lo sviluppo di un programma, il sistema operativo si occupa di controllare l'esecuzione del programma come pure le operazioni di ingresso/uscita, ad esempio la stampa del testo del programma o dei risultati dopo l'esecuzione. La capacità di gestione dei file consente al programma in fase di sviluppo di essere registrato (o caricato nella memoria per essere eseguito) utilizzando l'unità di memorizzazione su disco del computer. Queste caratteristiche del sistema operativo sono richieste dal software di sviluppo e possono essere usate anche dai programmi applicativi, se necessario. In quest'ultimo caso, i programmi applicativi sono progettati per essere eseguiti con un particolare sistema operativo.

Cross-software per lo sviluppo. Nella precedente discussione, era sottinteso che i programmi applicativi fossero sviluppati e destinati ad essere eseguiti sul medesimo sistema basato sull'MC68020. Un metodo alternativo consente che i programmi applicativi siano creati su un altro computer e convertiti in programmi eseguibili (in linguaggio-macchina) per l'MC68020. Questa tecnica, nota come sviluppo incrociato (*cross-development*), è possibile per i programmi in linguaggio assembler usando il cross-assemblatore (*Cross-Assembler*) della Motorola elencato nella Tab. 1.6. Sono disponibili varie versioni del Cross-Assembler, tra cui quelle per i minicomputer VAX™ * della Digital Equipment Corporation e per i personal computer dell'IBM. La Motorola fornisce anche i cross-compilatori per lo sviluppo di programmi in linguaggi ad alto livello.

Nell'utilizzare il software di sviluppo incrociato, il programmatore crea il programma nel linguaggio desiderato per un sistema di computer con una CPU MC68020, anche se l'esecuzione del compilatore o dell'assemblatore avviene su un altro computer denotato come "host" (ospite). Quindi, un cross-assemblatore in esecuzione sul computer host assembla le espressioni in linguaggio assembler dell'MC68020 e le converte in codice eseguibile o in una forma equivalente, leggibile dalla macchina, che potrebbe essere eseguita su un sistema basato sull'MC68020.

Il vantaggio del metodo dello sviluppo incrociato è che un programmatore il quale non disponga di un sistema di sviluppo basato sull'MC68020, come il SYS1131 della Motorola, può comunque avvalersi dei servizi di sviluppo del programma disponibili nel computer host. Tali servizi comprendono l'editor di testo per la preparazione del programma, un'unità di memoria a disco per la registrazione di programmi completi, ed un'unità stampante per stampare il programma ai fini della revisione o della documentazione.

Per eseguire un programma cross-assemblato, le istruzioni scritte in linguaggio-macchina possono essere elaborate da un programma *simulatore* o da un sistema di computer basato sull'MC68020. Con un simulatore, il funzionamento

* VAX è un marchio registrato di Digital Equipment Corporation.

dell'MC68020 viene simulato sul computer di sviluppo incrociato. Questo metodo è utile quando non è disponibile l'hardware del sistema MC68020 ma i programmi devono essere provati. In effetti, il simulatore fornisce un modello software del processore MC68020. Tuttavia, quegli aspetti che dipendono dall'hardware, come la temporizzazione, devono essere provati sul computer "target" (di destinazione) basato sull'MC68020. Ciò è fattibile trasferendo il programma cross-assemblato dal sistema di sviluppo alla memoria del sistema target.

Software per MC68020 da altri fornitori. In termini relativi, il numero di prodotti software forniti dalla Motorola è piccolo se paragonato al numero totale di sistemi operativi, programmi per lo sviluppo del software e programmi per applicazioni specifiche disponibili per i computer basati sull'MC68020. Alcune società indipendenti dalla Motorola forniscono un'estesa varietà di programmi che vengono eseguiti su sistemi basati sull'MC68020. Alcune di queste società producono anche cross-software per i personal computer più noti, per i minicomputer e per computer di dimensioni maggiori.

Esistono vari sistemi operativi con caratteristiche speciali, progettati per l'esecuzione su computer basati sull'MC68020. Sono disponibili sistemi operativi per operazioni in tempo reale (in cui il tempo è un fattore critico), per sistemi di memoria virtuale, o per computer multiprocessore. Alcuni di questi possono essere più efficienti od offrire più funzioni rispetto ai sistemi operativi disponibili dalla Motorola. In altri casi, il software può essere meno costoso per una particolare applicazione.

Possono essere acquistati compilatori per quasi ogni linguaggio di computer, tra cui Ada™*, C, COBOL, FORTH e PL/M, tanto per nominarne qualcuno. Sono disponibili cross-compilatori per la maggior parte dei computer più noti, come pure dei compilatori che possono operare direttamente su computer con una CPU MC68020. Anche assemblatori e cross-assemblatori sono forniti da varie organizzazioni indipendenti.

Ad esempio, la società Quelo, Inc. di Seattle, Washington, produce cross-assemblatori di MC68020 per i personal computer IBM e per il computer VAX. Questi assemblatori operano sotto il controllo del sistema operativo del computer host per produrre del codice eseguibile per l'MC68020. L'impiego di software della Quelo sarà discusso ulteriormente nel cap. 5. Un'altra società, la Big Bang Software, Inc., produce un programma simulatore per l'MC68020 che viene eseguito su personal computer IBM.

1.2.3 Sistemi di sviluppo

Gli elementi della famiglia dell'MC68020 descritti nei due sottoparagrafi precedenti possono essere combinati con i dispositivi periferici appropriati per creare un sistema di computer adatto ad un sistema di sviluppo del software. Il sistema

* Ada è un marchio registrato del Dipartimento della Difesa degli Stati Uniti d'America.

SYS1131, mostrato nella Fig. 1.7, è un sistema di questo tipo. Esso comprende un'unità a disco e può gestire fino a otto utenti. Una volta che i programmi applicativi sono stati progettati, essi possono essere creati, provati e corretti sul sistema di sviluppo. Se i programmi completi devono essere eseguiti sul sistema di sviluppo stesso, i test garantiscono che l'intero sistema soddisfa i requsiti dell'applicazione. Per esempio, con l'appropriato software applicativo, il SYS1131 potrebbe servire da sistema aziendale in grado di effettuare il controllo dell'inventario, il piano di lavoro ed altri compiti necessari. In un'altra configurazione, il sistema potrebbe fungere da controllore in un'applicazione industriale in cui il tempo è un fattore critico. Pertanto, il sistema di sviluppo ed il prodotto finale o computer "target" possono essere il medesimo sistema.

Quando è richiesta sia la progettazione che lo sviluppo dell'hardware per una certa applicazione, l'impegno di sviluppo deve includere l'integrazione del software applicativo con l'hardware del computer target. Quest'ultimo è di solito diverso dal computer scelto per il sistema di sviluppo dell'applicazione. L'integrazione consiste di una serie estesa di test dei componenti software che controllano l'hardware creato per il sistema target. In applicazioni tipiche, tale hardware può consistere di applicazioni speciali. Esso include la circuiteria d'interfaccia che non è disponibile come parte della famiglia di chip d'interfacciamento della Motorola. Quando si utilizzano i dispositivi della famiglia, si può scegliere per il computer target un sistema operativo che includa i programmi per il controllo dell'hardware in modo da rendere necessario solo un minimo impegno di sviluppo e di test. Se nessun sistema operativo dev'essere incluso nel sistema target, allora lo sviluppo dei programmi d'interfacciamento per i chip della famiglia dev'essere creato o acquistato ed integrato nel sistema target. Questo caso si presenta spesso quando il target è un prodotto quale uno strumento di laboratorio che esegue un'elaborazione di computer.

Il ciclo di produzione del software per lo sviluppo di programmi che richiedono l'integrazione di hardware e software è mostrato in forma semplificata nella Fig. 1.9. Se i programmi sono eseguiti correttamente sul sistema di sviluppo, per quanto concerne i loro test su di esso, allora essi potranno essere trasferiti al computer target per ulteriori prove con l'hardware effettivo del prodotto finito. Gli eventuali errori saranno quindi corretti finché il prodotto non funzionerà come richiesto.

La Fig. 1.10 illustra una stazione di sviluppo che può essere connessa tra un computer a piastra singola (computer target) ed il sistema di sviluppo. Per l'integrazione hardware/software, l'esecuzione del programma sul sistema target viene controllata col sistema di sviluppo tramite i comandi appropriati inseriti dall'utente. Questi comandi consentono il collaudo e la correzione ("debugging") dell'intero sistema target, in maniera molto simile a quella con cui il software di debugging già discusso permetteva la messa a punto dei programmi eseguiti sul sistema di sviluppo stesso. Per esempio, sul sistema target possono essere eseguite singole istruzioni o una piccola porzione del programma; quindi possono essere visualizzate per l'utente le informazioni relative al debugging.

Fig. 1.9
Produzione di software.

Definizione del problema
Progetto
Selezione del linguaggio
Progetto
Codifica del programma
Debugging e test
Documentaz.
Sviluppo
Eventuale correzione
Esecuzione su sistema target
Integrazione hardware software

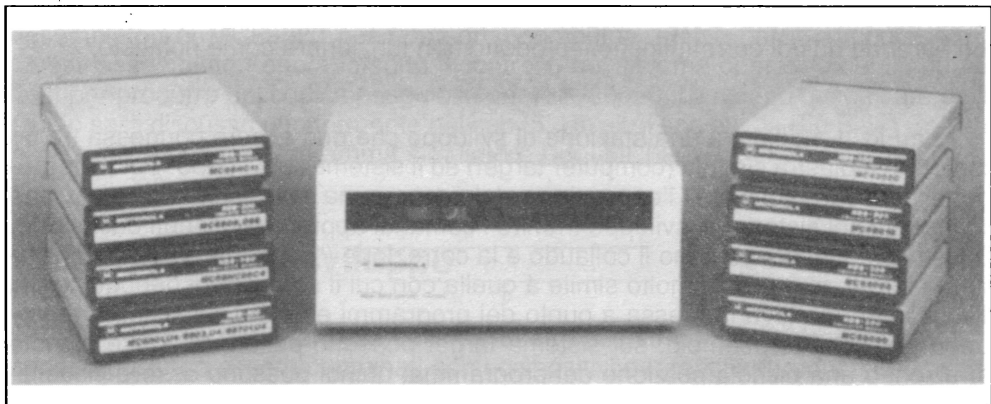
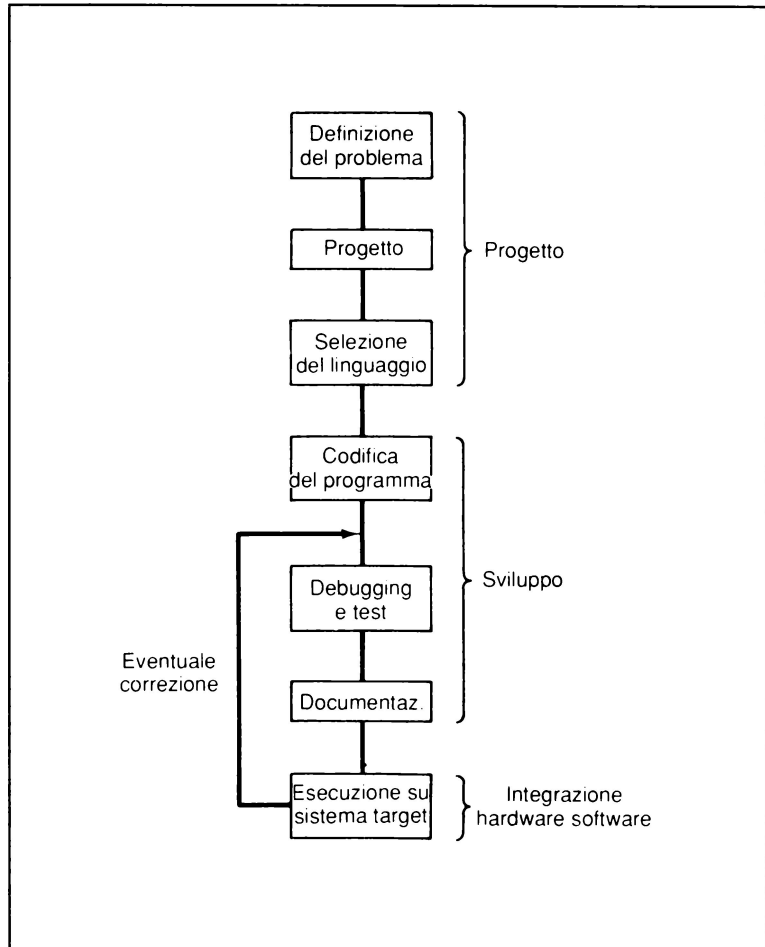



Fig. 1.10 *Sistema di sviluppo della Motorola per l'integrazione di hardware e software. (Per gentile concessione di Motorola, Inc.)*

1.2.4 Moduli di computer a livello di piastra

I "chip" discussi nel par. 1.2.1 formano l'elemento fondamentale, insieme con altri circuiti necessari, per l'implementazione hardware di un sistema di computer. La CPU, i chip d'interfacciamento ed i chip di memoria sono elettricamente connessi su una o più piastre circuitali per creare l'hardware del sistema di elaborazione e di memorizzazione. Volendo, un produttore di un sistema di computer o di un prodotto, il quale disponga degli opportuni mezzi tecnici e di produzione, può progettare, collaudare e realizzare le piastre complete, acquistando solamente i chip da un fornitore come la Motorola. Comunque, in molti casi, un altro metodo è quello di acquistare i moduli a livello di piastra dalla Motorola o da altri fornitori; allora il produttore dovrà soltanto assemblare il prodotto usando i moduli indicati per l'applicazione.

La Fig. 1.11 mostra un sistema di computer assemblato dai moduli a livello di piastra VERSAbus della Motorola. Un sistema di questo tipo conterrà di solito un computer a piastra singola come unità di elaborazione, oltre che moduli contenenti la memoria ed eventualmente altri moduli per controllare i dispositivi periferici. I sistemi di computer modulari completi che impiegano i moduli VMEbus saranno descritti nel cap. 14.

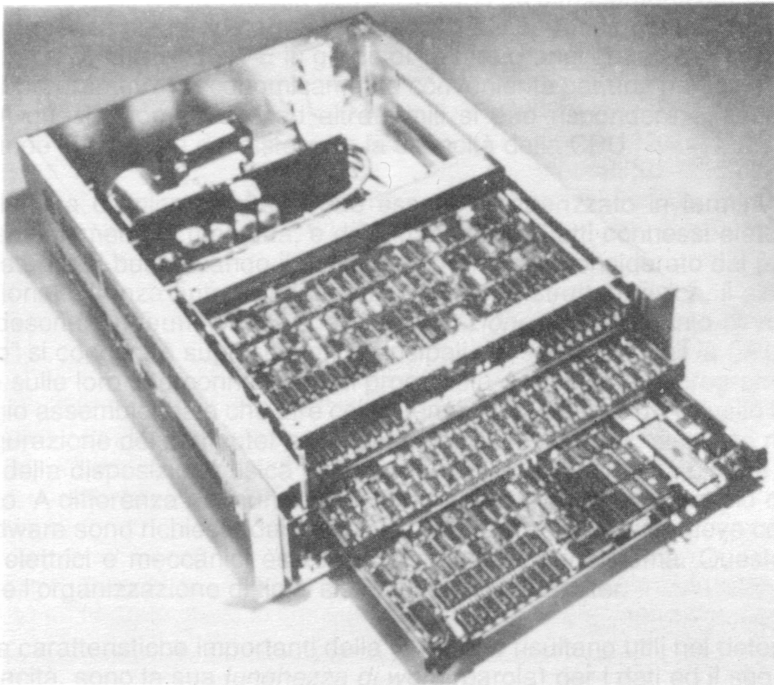



Fig. 1.11 *Costruzione modulare di un sistema di computer. (Per gentile concessione di Motorola, Inc.)*

CAPITOLO 2

CARATTERISTICHE DEI MICROCOMPUTER E DEI MICROPROCESSORI

I sistemi di microcomputer operano su sequenze di cifre binarie (*binary digit:* bit), che rappresentano istruzioni in codice-macchina destinate ad essere eseguite dalla CPU oppure un valore di dati da trasferire, trasformare o trattare in qualche altro modo. La facilità e la velocità con cui la CPU ed altri elementi del sistema trattano le sequenze binarie determina le caratteristiche fondamentali del sistema di computer. La CPU è veloce? È in grado di gestire grandi quantità di dati come pure grandi programmi? È economicamente conveniente per una particolare applicazione? A queste domande e ad altre simili si può rispondere almeno in parte esaminando la struttura del sistema e la capacità della CPU.

Il sistema di microcomputer può essere caratterizzato in termini della sua CPU, degli elementi di memoria, e dei circuiti di I/O, tutti connessi elettricamente da un sistema di bus. Quando il microcomputer viene considerato dal punto di vista funzionale, senza entrare nei dettagli della sua struttura fisica, il sistema può essere descritto in termini della sua *organizzazione*. Questo punto di vista "organizzativo" si concentra sugli elementi principali del sistema, quali la CPU e la memoria, e sulle loro interconnessioni. Il progettista di sistema o il programmatore in linguaggio assembler ha a che fare col sistema a questo livello. Al livello hardware, la configurazione del computer su singola piastra MVME133, discusso nel cap. 1, è tipica della disposizione fisica dei componenti su una singola piastra a circuito stampato. A differenza del punto di vista organizzativo, descrizioni più dettagliate dell'hardware sono richieste dal progettista d'interfaccia, il quale deve conoscere i dettagli elettrici e meccanici esatti dei componenti del sistema. Questo capitolo concerne l'organizzazione di tipici sistemi di microcomputer.

Due caratteristiche importanti della CPU, che risultano utili nel determinare la sua capacità, sono la sua *lunghezza di word* (parola) per i dati ed il suo *campo di indirizzamento*. La lunghezza di word si riferisce al numero di bit in un elemento di dati che possono essere trasferiti con una singola operazione (in parallelo) tra la CPU ed altri elementi del sistema. Il campo d'indirizzamento definisce il numero di

locazioni di memoria indirizzabili dalla CPU. Per esempio, l'MC68020 della Motorola ha una lunghezza di word di 32 bit ed un campo d'indirizzamento di oltre 4 miliardi di locazioni di memoria.

In questo capitolo sono definite le più comuni organizzazioni di microcomputer e vengono descritti gli elementi principali di un sistema di microcomputer. Dopodiché saranno illustrate le differenze tra processori tipici a 8, 16 e 32 bit, descrivendone le lunghezze di word ed i campi d'indirizzamento. Queste descrizioni riguardano sistemi utilizzati per eseguire le funzioni fondamentali di un computer, come l'esecuzione delle istruzioni e le operazioni di ingresso/uscita. Le informazioni fornite in questi primi due paragrafi servono per presentare al lettore il microprocessore nel suo ruolo di CPU. Sarà inoltre sottolineata l'interazione tra le varie componenti del computer.

Nel paragrafo finale di questo capitolo saranno presentati tre punti di vista del microcomputer, a cui seguirà un'introduzione delle caratteristiche dell'MC68020 e di microprocessori simili. Questi punti di vista sono quelli del progettista di sistemi, del programmatore in linguaggio assembler e del progettista d'interfaccia. L'MC68020 è stato sviluppato per soddisfare tre esigenze: come CPU in un sistema di computer, come processore programmabile e come chip hardware.

2.1 ORGANIZZAZIONE DEL MICROCOMPUTER E STRUTTURA DEL BUS

Gli elementi di un semplice microcomputer sono mostrati in Fig. 2.1. Questo diagramma a blocchi mostra il microprocessore (CPU), un'unità di memoria e la circuiteria di I/O. La CPU comunica con gli altri elementi del sistema tramite linee di segnale parallele che, nel complesso, costituiscono il bus interno di sistema.[1] Nella Fig. 2.1, il bus è mostrato suddiviso funzionamente in linee d'*indirizzo*, linee di *dati* e linee di *controllo*. Queste linee di segnali servono a connettere elettricamente la CPU ai vari circuiti, al fine di trasferire i dati dentro e fuori dal microprocessore, come pure a connettere un'unità di memoria contenente le istruzioni e i dati per il processore.

La Fig. 2.2 illustra un diagramma più dettagliato, che mostra le connessioni della CPU MC68020 coi suoi coprocessori. Nella figura, l'MC68020 è mostrato con una memoria "cache" sul proprio chip. In questo tipo di memoria sono registrate le istruzioni che la CPU ha prelevato dalla memoria principale. Se la CPU determina che un'istruzione richiesta si trova nel cache, allora essa non sarà prelevata nuovamente dalla memoria principale, risparmiando così il tempo di un accesso alla memoria esterna. La capacità della memoria cache è molto limitata rispetto a quella della memoria principale e la sua utilità massima si rivela quando un programma

[1] Le linee di segnale parallele indicano che ogni linea di segnale può essere usata simultaneamente con le altre e indipendentemente per trasferire i segnali elettrici che rappresentano l'informazione. Queste 32 linee di segnali di dati consentono il trasferimento simultaneo di 32 bit. Per contro, una linea seriale di dati richiederebbe 32 trasferimenti di un solo bit alla volta per svolgere la medesima funzione.

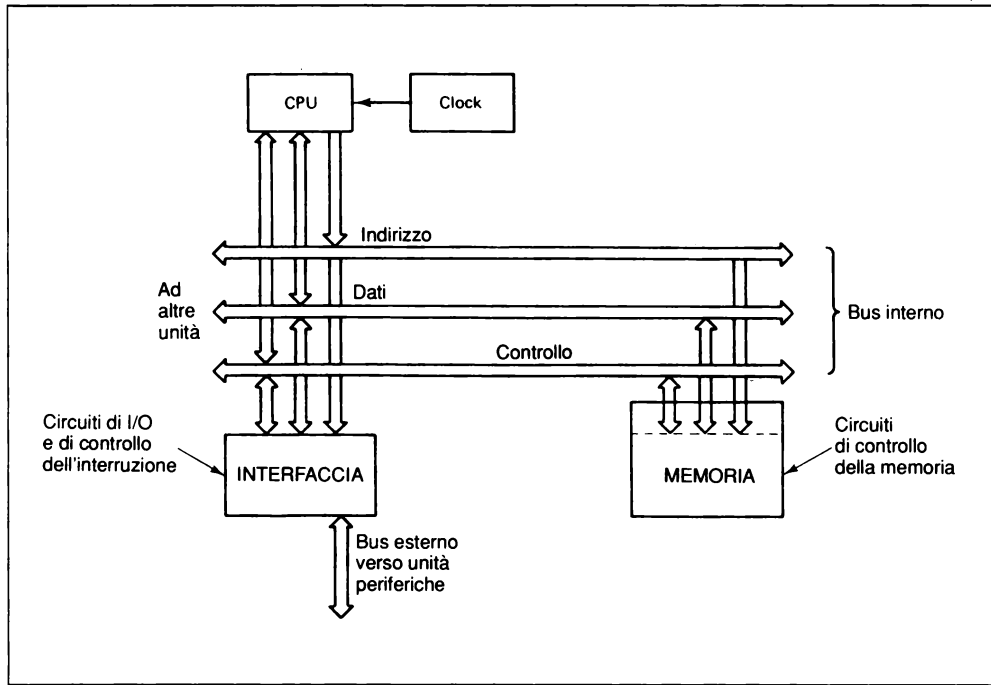



Fig. 2.1 *Organizzazione semplificata di un microcomputer.*

Nota: La memoria cache contiene soltanto 256 byte di istruzioni. La memoria principale può avere fino a 4 gigabyte (4294697296 byte) di memoria per memorizzare sia istruzioni che operandi. La CPU ed i coprocessori sono interconnessi da un bus che ha in totale 114 linee di segnale, tra cui quelle per l'alimentazione elettrica del chip.

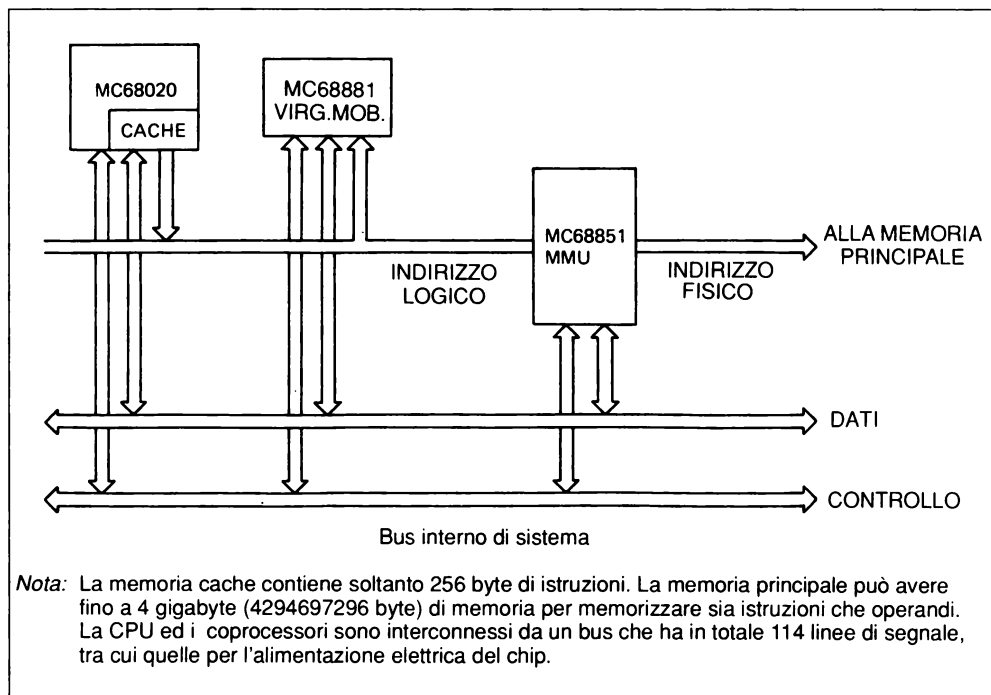



Fig. 2.2 *L'MC68020 con i suoi coprocessori.*

fa sì che la CPU esegua un ciclo ripetitivo di una breve sequenza di istruzioni. In questo caso, una volta che il cache è stato riempito con la sequenza di istruzioni durante la prima esecuzione del ciclo, la CPU dovrà solamente leggere o scrivere gli operandi di dati nella memoria senza dover prelevare nuovamente le istruzioni nei cicli successivi.

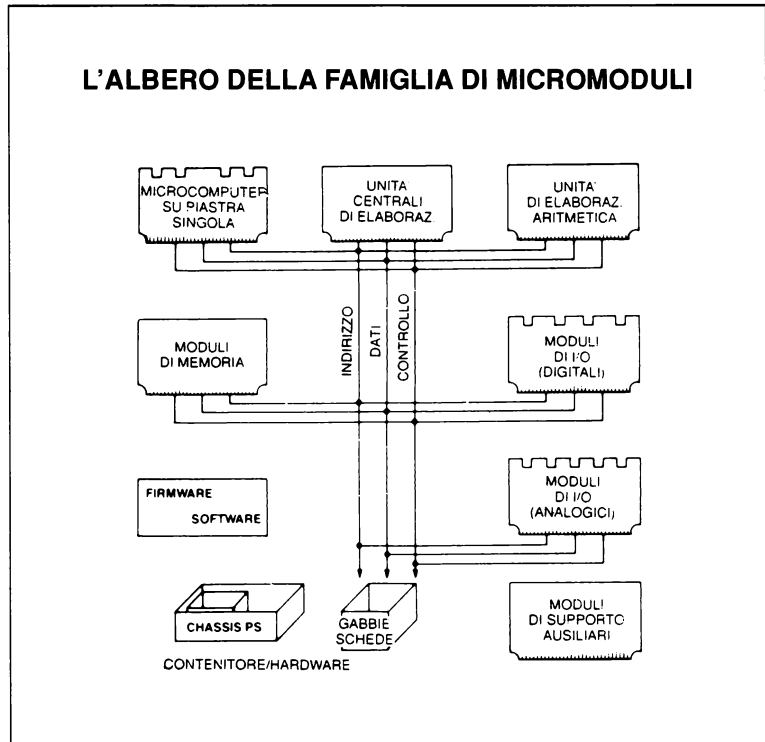
Il coprocessore in virgola mobile esegue le operazioni aritmetiche in risposta a speciali istruzioni eseguibili, codificate per indicare che l'oggetto dell'indirizzamento è il coprocessore. Si noti che quest'ultimo non fa parte dello spazio di memoria del sistema; tuttavia le istruzioni del coprocessore e gli operandi di dati sono tenuti nella memoria principale. Nella Fig. 2.2 è mostrata anche un'unità di gestione della memoria (*Memory Management Unit:* MMU), che converte gli indirizzi logici di un programma in indirizzi fisici nella memoria, come discusso nel par. 12.4.

Sebbene la memoria cache ed i coprocessori aggiungano una certa complessità alla comprensione dettagliata dei computer basati sull'MC68020, un programmatore in linguaggio assembler può considerare anche il sistema più complicato come se avesse un'organizzazione simile a quella del computer semplificato della Fig. 2.1. Questo punto di vista è corretto per la maggioranza dei programmi applicativi, poiché le attività del cache e dell'unità di gestione della memoria sono automatiche, una volta che il sistema operativo ha provveduto alla loro inizializzazione. Usando il coprocessore in virgola mobile, il programmatore vede l'insieme di istruzioni del coprocessore come un'estensione dell'insieme di istruzioni della CPU. Nessun'altra informazione sul coprocessore è necessaria al programmatore che crea del software applicativo. Questo punto di vista semplificato di un sistema basato sull'MC68020 è adeguato in questo libro finché non saranno presentati i dettagli della memoria cache e dei coprocessori nel cap. 12. Pertanto, al fine di comprendere il funzionamento del sistema completo per varie applicazioni, la CPU nella Fig. 2.1 potrebbe essere considerata come se fosse composta dall'insieme di chip MC68020/MC68881/MC68851.

Il sistema nella Fig. 2.1 può essere costruito in vari modi, utilizzando i componenti della famiglia di microprocessori descritti nel cap. 1. Al livello più elementare, i singoli chip di circuiti integrati per la CPU, gli elementi di memoria e l'I/O possono essere combinati da un progettista hardware che sia a conoscenza delle caratteristiche elettriche di ciascun elemento. Un approccio a più alto livello, che rappresenta il progetto a livello di piastra, si effettua combinando un computer su piastra singola con varie altre piastre circuitali contenenti sottosistemi di memoria e sottosistemi di I/O. La costruzione di microcomputer a partire da moduli quali i VME della Motorola segue questo metodo di progettazione a livello di piastra. Un modulo della specie utilizzata in questo tipo di progetto era stato mostrato nella Fig. 1.3.

La Fig. 2.3 illustra una possibile struttura di un sistema a livello di piastra. I moduli vengono "innestati" nei connettori in un telaio che contiene la struttura del bus ed un alimentatore. Ulteriori moduli, come un'unità aritmetica per applicazioni scientifiche o moduli di I/O per scopi speciali, possono essere aggiunti per espandere il sistema. L'organizzazione è definita dal progettista del sistema, che specifica le capacità di memoria e di I/O necessarie per soddisfare i requisiti della particolare applicazione.

Fig. 2.3 *Implementazione di un microcomputer su piastra singola. (Per gentile concessione di Motorola, Inc.)*

L'ALBERO DELLA FAMIGLIA DI MICROMODULI

Il sistema hardware completo potrebbe servire come base per una certa varietà di prodotti, da una macchina per saldatura automatica fino ad un computer "general purpose". Il software sarebbe sviluppato per soddisfare le esigenze specifiche. Una porzione del software in linguaggio-macchina potrebbe essere memorizzata in una memoria a sola lettura, in modo da non poter essere alterata dopo che il sistema è stato completato. Tale codice è talvolta denominato *firmware*, come indicato nella Fig. 2.3 Nei prossimi capitoli saranno esplorate alcune differenze tra il progetto del software per le memorie di lettura/scrittura e per quelle a sola lettura.

In un sistema a livello di piastra, possono essere richieste interfacce ad unità periferiche speciali. Poiché il produttore degli altri moduli non può fornire l'interfaccia unica richiesta, un progettista d'interfaccia potrebbe svilupparne una "su misura". L'integrazione delle routine hardware e software per queste interfacce speciali è una parte d'importanza capitale nello sviluppo di un sistema.

2.1.1 L'unità centrale di elaborazione e il clock

La CPU controlla e coordina tutte le attività nel microcomputer. Essa esegue istruzioni in linguaggio-macchina, prelevate dall'unità di memoria, ed effettua tutte le operazioni aritmetiche, logiche o di altro tipo richieste dalle istruzioni. La CPU

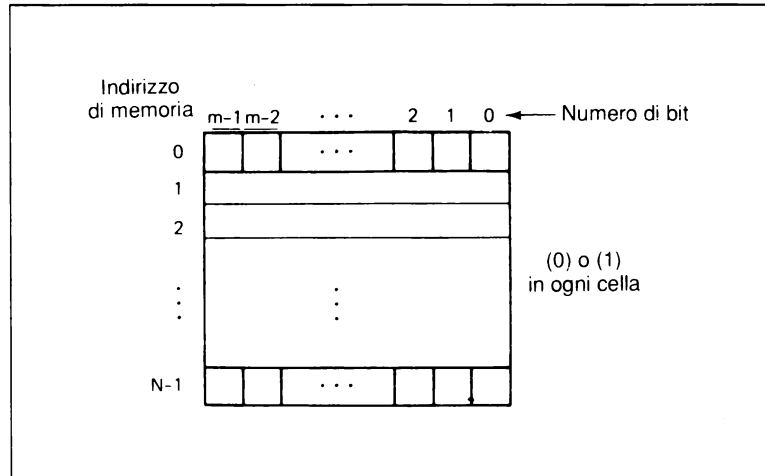
può *leggere* valori di dati (operandi) dalla memoria o scrivere valori nella memoria, inviando gli appropriati segnali elettrici (comandi) tramite il bus interno. La CPU può anche iniziare un'operazione di I/O per il trasferimento di dati da/verso dispositivi periferici. Il bus esterno mostrato in Fig. 2.1 connette il sistema ad una o più unità periferiche e consente alla CPU di svolgere tali operazioni di I/O.

L'occorrenza di eventi è coordinata con precisione dal clock di sistema. Il clock è un "orologio" che scandisce il passare del tempo ad intervalli di qualche decimilionesimo di secondo o anche più brevi. In ciascun intervallo, la CPU o uno degli altri componenti del sistema svolge una precisa funzione, come la presentazione di un valore di dati sulle linee di segnali di dati. La frequenza del clock (cioè il numero di "ticchettii" al secondo) determina la velocità operativa fondamentale del sistema. Se la frequenza di clock raddoppia, dovrebbe raddoppiare anche la velocità operativa del sistema, a meno che non intervengano altri fattori — come il tempo di risposta della memoria — a limitare la velocità. La massima frequenza di clock per il sistema è determinata dalla CPU. Sono disponibili versioni dell'MC68020 che presentano differenze nella massima frequenza di clock, come già discusso nel cap. 1. Nel cap. 13 sarà fornita una definizione più precisa del clock di sistema per sistemi basati sull'MC68020.

Molti computer hanno un clock che determina la velocità operativa di ciascun componente e chip elettronico connesso al suo bus di sistema interno. Non è necessario che un computer basato sull'MC68020 abbia un singolo clock che sincronizza ogni elemento del sistema. Per esempio, il coprocessore in virgola mobile o i circuiti di controllo della memoria possono operare a frequenze di clock differenti da quella della CPU. Queste unità potrebbero utilizzare clock indipendenti da quello della CPU. Le linee di segnali di controllo del bus interno di sistema sono utilizzate per coordinare il trasferimento di dati da e verso la CPU alla velocità o frequenza richiesta da ciascun elemento del sistema. Ciò è noto come trasferimento asincrono di dati e serve a consentire la connessione di unità periferiche relativamente più lente o più veloci al bus della CPU, al fine di garantire l'affidabilità delle operazioni.

2.1.2 L'unità di memoria

Nel sistema con memoria singola mostrato in Fig. 2.1, l'unità di memoria contiene sia le istruzioni che i dati che saranno utilizzati dalla CPU. Ciascuna cella di memoria nell'unità suddetta contiene un solo bit e la loro organizzazione è illustrata in Fig. 2.4. Ogni locazione di memoria contenente *m* bit è individuata da un numero positivo, il suo *indirizzo*, che indica la posizione di tale locazione nella memoria. La CPU può fare riferimento ad una singola locazione di memoria tramite le linee di segnali d'indirizzo e può controllare le operazioni della memoria mediante linee di controllo selezionate. La memoria stessa è organizzata in unità d'informazione, le cosiddette *word*. La dimensione di una word può variare da 8 bit — che costituisce ciò che viene definito *byte* nell'ambito dei microcomputer — fino a 64 bit o più nei grandi computer. Per comodità, nel confronto delle capacità possedute dalle varie memorie nel registrare le informazioni, una word è di solito

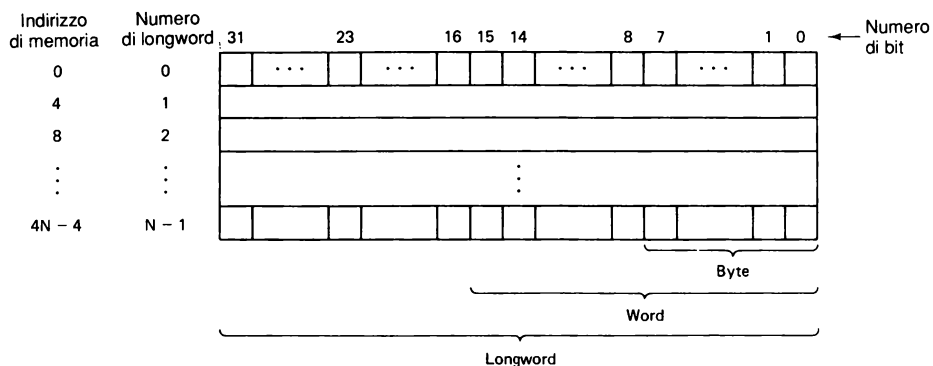
Fig. 2.4
Organizzazione della memoria, per una memoria di m bit consistente di N locazioni.

suddivisa in un certo numero di byte. Per esempio, la word di memoria dell'MC68020 è definita come composta da 16 bit o due byte, come mostrato in Fig. 2.5. L'MC68020 è una CPU *indirizzabile per byte*, poiché può indirizzare l'uno o l'altro dei due byte (i bit 0-7 o i bit 8-15) di una word di memoria. Essa può indirizzare anche locazioni di word, consistenti di due byte, come pure longword (word lunghe) costituite da quattro byte.

Il *contenuto* di quasiasi locazione di memoria può essere ottenuto dalla memoria sulle linee dei segnali di dati (la CPU legge), oppure il valore delle linee dei segnali di dati può essere memorizzato nella locazione indirizzata (la CPU scrive). L'attività effettiva dell'unità di memoria è diretta dai circuiti di controllo della memoria, non mostrati nelle Figg. 2.4 e 2.5. La lunghezza della memoria, specificata come N locazioni in Fig. 2.4, è normalmente una potenza di 2, come 1024 o 4096. In generale, il numero di locazioni è $2k = N$, dove k è un intero. L'MC68020 impiega 32 bit per rappresentare un indirizzo. Pertanto l'MC68020 può indirizzare 2^{32} distinte locazioni di memoria, ciascuna contenente un byte d'informazione. La sua capacità d'indirizzamento di word è quindi di 2^{31} word di 16 bit. La lunghezza della memoria è spesso specificata come un multiplo di 1024 byte, che viene denotato come 1 KB (kilobyte), o di 1048576 byte, noto come 1 MB (megabyte).[2]

Il numero di bit che possono essere trasferiti contemporaneamente (in parallelo) è determinato dal numero di linee di segnali di dati che connettono la memoria e la CPU. Un tipico microcomputer a 8 bit ha otto linee di dati e la memoria è organizzata in byte. L'MC68000 ha un cammino di dati largo 16 bit, che consente un trasferimento di byte su otto linee o un trasferimento di due byte (16 bit) sulle 16 linee di segnale. Invece l'MC68020 a 32 bit può trasferire 8, 16 o 32 bit in una sola volta.

[2] A rigor di termini, i prefissi kilo, mega e giga significano 10^3, 10^6 e 10^9, rispettivamente. Quando si fa riferimento alla capacità di memoria, il loro significato è spesso quello di 2^{10}, 2^{20} e 2^{30}, rispettivamente.

Indirizzo di memoria | Numero di longword | 31 ... 23 ... 16 15 14 ... 8 7 ... 1 0 ← Numero di bit

0 0
4 1
8 2
4N − 4 N − 1

Byte
Word
Longword

(a) *Organizzazione per longword.*

Indirizzo di memoria | Numero di word | 15 14 ... 8 7 ... 1 0 ← Numero di bit

0 0
2 1
4 2
4N − 2 2N − 1

(b) *Organizzazione per word.*

Indirizzo di memoria | Numero di byte | 7 ... 1 0 ← Numero di bit

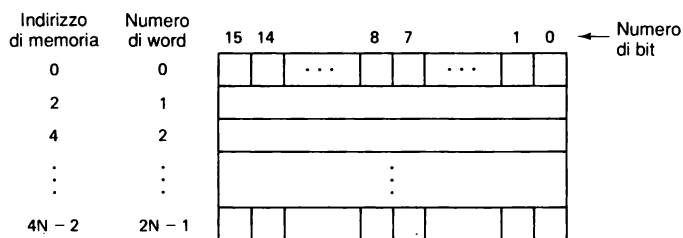
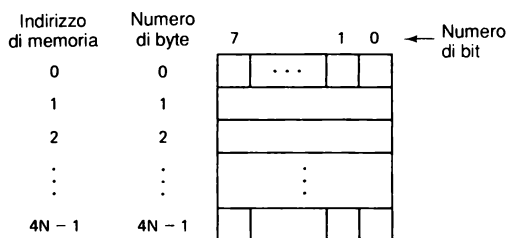
0 0
1 1
2 2
4N − 1 4N − 1

(c) *Organizzazione per byte.*

Fig. 2.5 *Organizzazione della memoria in un sistema basato sull'MC68020 avente 4N byte di capacità di memorizzazione.*

Un aspetto della flessibilità dei computer basati sull'MC68020 deriva appunto dalla loro capacità di operare su operandi di byte, word o longword secondo le istruzioni del programma. Considerando gli indirizzi della memoria fisica come mostrato nella Fig. 2.5, l'indirizzo di un byte può essere un numero pari o dispari, l'indirizzo di una word è un qualsiasi numero pari, mentre l'indirizzo di una longword è un numero pari multiplo di 4. Il programmatore specifica la lunghezza dell'operando come un byte, una word o una longword in qualsiasi istruzione dell'MC68020 che fa riferimento a tale operando. Se il programmatore definisce l'operando nella memoria in modo che gli operandi di word o di longword siano memorizzati in locazioni fisiche di word o di longword, allora l'operando viene definito "allineato" (cioè, allineato con la memoria fisica). Comunque, un operando di word o di longword può iniziare da qualsiasi indirizzo, pari o dispari, secondo quanto stabilito dal programmatore. L'indirizzamento della memoria con l'MC68020 sarà spiegato ulteriormente nel par. 4.6.

Nella Fig. 2.6 è illustrata una possibile organizzazione di un tipico prodotto basato su microcomputer. In questo esempio, si sottintende che il sistema operativo ed i programmi applicativi siano memorizzati in una ROM. La memoria di lettura/scrittura contiene valori che saranno probabilmente soggetti a modifiche nella fase di attività del prodotto. Tale memoria è di solito definita "ad accesso casuale" (*Random Access Memory:* RAM), per distinguerla dalla memoria a sola lettura (*Read Only Memory:* ROM) del computer. Un metodo alternativo, adottato nella maggior parte dei sistemi "general purpose", consiste nell'immagazzinare la massa dei programmi su un'unità a disco esterna e di caricare i programmi nella memoria all'occorrenza. In questo caso, l'utilizzazione del disco è controllata dal sistema operativo.

Si noti che il sistema operativo e le sue routine per gestire l'I/O e le interruzioni sono separati dai programmi applicativi e dalla loro area di memoria di lettura/scrittura. Se un programma applicativo richiede il trasferimento di dati ad un'unità esterna, il trasferimento è controllato solitamente dal sistema operativo, per garantire la correttezza dello svolgimento delle operazioni del sistema.

2.1.3 Ingresso/uscita

La circuiteria di I/O mostrata in Fig. 2.6 è controllata dalla CPU per consentire i trasferimenti di dati tra il bus interno e il bus esterno che connette i dispositivi periferici al sistema. Tale circuiteria d'interfacciamento è progettata per soddisfare le esigenze del dispositivo periferico. Inoltre, l'attività del bus esterno è in genere indipendente dall'attività della CPU in termini di velocità e di larghezza del percorso di dati. Ad esempio, un terminale di operatore richiede normalmente il trasferimento di bit di dati in modo seriale, ad una velocità che è molto lenta rispetto a quella con cui i valori dei dati possono essere trasferiti in parallelo sul bus interno di sistema. La principale funzione della circuiteria di I/O in questo caso è quella di risolvere questa discordanza di velocità e formato. Non sono mostrate le linee di controllo delle interruzioni; tali linee avvisano il processore la condizione in cui il dispositivo è pronto.

Bus interno
Memoria*
CPU
Circuiti di I/O (interfaccia)
Programmi del sistema operativo
Routine di I/O e routine d'interruzione
Memoria ROM
Routine per controllare le operazioni esterne
Unità CRT, unità a disco ecc.
Unità periferiche speciali
Programmi applicativi
Memoria ROM

* Le aree di memoria non designate come ROM sono aree di lettura/scrittura.

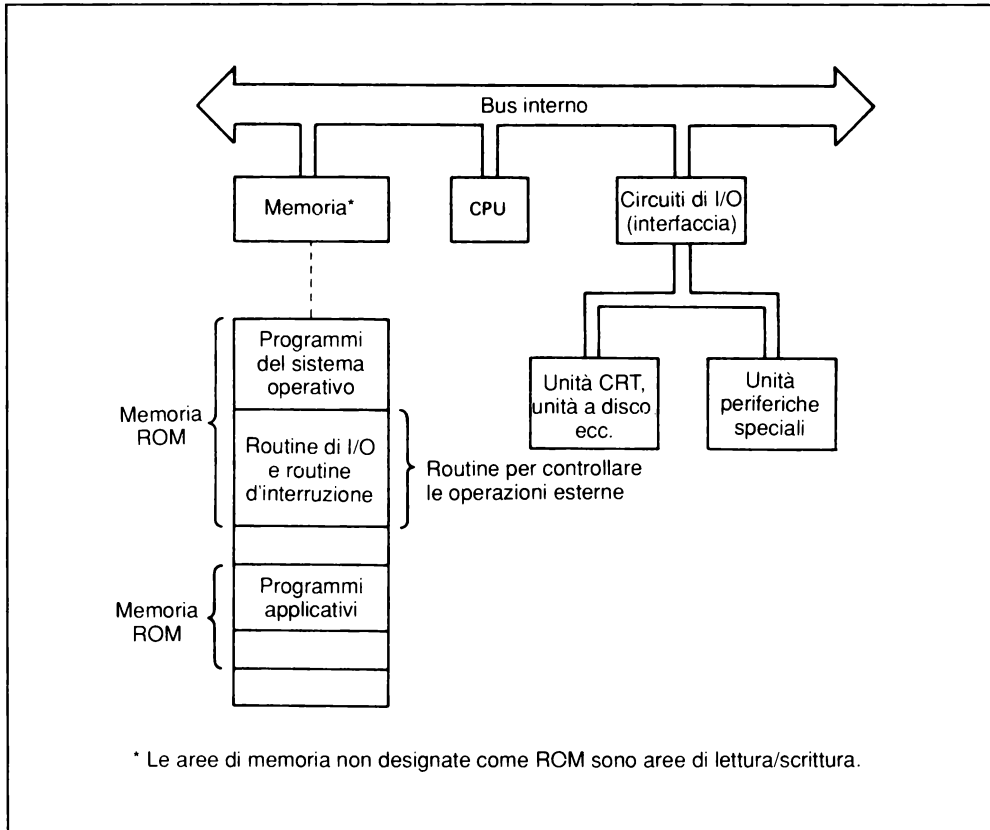



Fig. 2.6 *Organizzazione dell'hardware e del software per un prodotto basato su un microcomputer.*

I dispositivi periferici del microcomputer mostrati nella figura sono indirizzati dalla CPU nello stesso modo in cui viene indirizzata la memoria, poiché l'unità di memoria e l'unità di I/O sono effettivamente in parallelo sul medesimo bus. Si dice che i sistemi progettati in questa maniera hanno un I/O *rappresentato nella memoria* (*memory-mapped I/O*) poiché la CPU accede alla memoria ed ai circuiti di I/O basandosi soltanto su tali indirizzi. Questo metodo sarà discusso ulteriormente nel cap. 13, che tratterà la capacità d'ingresso/uscita dell'MC68020.

2.1.4 Il bus interno

L'architettura dei microcomputer discussi in questo paragrafo viene definita *a singolo bus*, poiché le unità di memoria e quelle di I/O condividono il medesimo bus con la CPU. Questo concetto fu introdotto dalla Digital Equipment Corporation con la sua famiglia di minicomputer PDP-11 ed è impiegato estesamente nei sistemi di microcomputer. Il numero di linee di segnali dedicate ad indirizzi e dati ed il numero di linee di segnali di controllo determinano essenzialmente la capacità del

sistema a singolo bus, poiché tutti i trasferimenti di dati o di istruzioni si presentano su questi percorsi di segnali. Ciascuna linea di segnale può assumere due stati; dunque n linee considerate in parallelo rappresentano 2^n stati in una word di n bit. L'MC68020 dispone effettivamente di 114 linee di segnali per gestire indirizzi e dati e segnali di controllo, oltre che parecchie altre funzioni.

ESERCIZI

2.1.1 Si confronti il progetto a livello di piastra col progetto a livello di chip di un prodotto basato su microprocessore. Si includano le considerazioni di natura tecnica e finanziaria. Per rispondere alla domanda, si consiglia di far riferimento alla letteratura tecnica delle case produttrici.

2.1.2 La CPU MC68020RC12 (12.5 MHz) può eseguire in 80 ns un'istruzione per azzerare un registro. Si calcoli il tempo di esecuzione di tale dalle CPU:
(a) MC6802RC16 (16.67 MHz)
(b) MC68020RC25 (25 MHz).

2.13 Qual è l'intervallo d'indirizzamento in esadecimale ed il numero di linee di segnali d'indirizzo necessari per indirizzare 1 megabute di memoria?

2.1.4 Si discutano alcune applicazioni in cui il microprocessore può non essere la migliore CPU per un prodotto o computer. Si considerino metodi che impiegano altri circuiti e chip elettronici, come pure i casi in cui è desiderabile un sistema di computer più grande.

2.2 LUNGHEZZA DI WORD E INTERVALLO D'INDIRIZZAMENTO DEL MICROPROCESSORE

I processori come l'MC68020 sono classificati come microprocessori a 32 bit poiché hanno 32 linee di segnali di dati per i trasferimenti dei dati. Per i microprocessori attualmente disponibili in questa classe, il numero di linee d'indirizzamento è tipicamente 32. Le 32 linee d'indirizzo dell'MC68020 consentono al processore d'indirizzare oltre 4 miliardi di locazioni di memoria da un byte ciascuna. In questo paragrafo saranno discusse le capacità e le caratteristiche dei microprocessori in base alla loro lunghezza di word ed al loro intervallo d'indirizzamento.

2.2.1 Lunghezza di word

Per definire la *lunghezza di word* di un microcomputer s'impiega spesso la lunghezza della word di dati più comunemente usata. Sebbene possano essere adottate anche altre definizioni, il *computer a m bit* è definito qui come un computer i cui percorsi principali di dati esterni al processore sono percorsi paralleli di m bit. Quindi, essendo un numero intero, la word di m bit trasferita tra il processore ed altri elementi del sistema, come la memoria, può rappresentare 2^m numeri nell'intervallo decimale da 0 a $2^m - 1$. Una lunghezza di word di 8 bit consente di

definire soltanto 256 valori nell'intervallo da 0 a 255. La word di dati di 16 bit dell'MC68000 consente 65536 valori. La longword di 32 bit dell'MC68020 permette 4294967296 valori.

La significatività della lunghezza di word appare evidente esaminando l'intervallo numerico. In generale, il progetto, la funzione desiderata e l'efficienza di un processore possono essere desunte dalla lunghezza della word, almeno per i processori di finalità generale. Tutte le informazioni trasferite verso e dal processore — incluse le sequenze di codici binari che rappresentano le istruzioni — hanno lunghezze che sono multipli o sottomultipli della word di dati di *m* bit. Poiché queste quantità di *m* bit consentono di assegnare alla configurazione di bit 2^m combinazioni distinte, è possibile codificare in qualche modo 2^m istruzioni, valori di dati o altre entità. Di per sé, la lunghezza di *m* bit non è una limitazione severa. Tramite un'opportuna programmazione, una quantità può essere rappresentata come due o più valori di *m* bit combinati per formare entità lunghe più word. Un processore a 8 bit può operare generalmente su quantità di 8 bit, 16 bit o 32 bit, trattando 8 bit alla volta. La penalità per operazioni su più word è una riduzione della velocità operativa rispetto ad operazioni che usano esclusivamente quantità di 8 bit. Un processore a 16 bit può ovviamente trasferire quantità di 16 bit con una velocità almeno doppia di quella di un'unità a 8 bit che deve utilizzare due cicli di trasferimento, a parità di ogni altra condizione. Nell'ambito dei microprocessori, le lunghezze di word di 4, 8, 16 e 32 bit servono ad operare una classificazione. L'MC68008 descritto nel cap. 1 è più difficile da classificare in questo schema poiché ha otto linee di dati ma per il resto si comporta come l'MC68000 con 16 linee di dati.

2.2.2 Intervallo d'indirizzamento

Un'altra caratteristica di un microprocessore che ne determina le capacità è il suo *intervallo d'indirizzamento*. Qualsiasi elemento (istruzione o dati) trasferito lungo le linee di dati di un processore è invidiato da un indirizzo che ne identifica la posizione esatta nella memoria. L'intervallo d'indirizzamento determina la dimensione del programma più grande o il massimo numero di valori di dati che il processore può indirizzare. Se i cammini di segnale per un indirizzo sono paralleli di *k* bit, allora il processore può indirizzare 2^k locazioni distinte.

Come affermato in precedenza, l'MC68020 può indirizzare 2^{32} locazioni di bit. Tranne che per i microprocessori a 32 bit, la lunghezza dell'indirizzo di *k* bit è maggiore della lunghezza della word di dati di *m* bit per la maggior parte dei processori. Per esempio, l'MC68000 ha $k = 24$ e $m = 16$. Un indirizzo di 32 bit, come l'MC68020, è considerato sufficiente per quasi tutte le applicazioni, poiché la CPU potrebbe indirizzare oltre 4 *miliardi* di locazioni. Molti processori a 8 bit impiegano una lunghezza d'indirizzo di 16 bit, che consente d'individuare un numero di locazioni distinte pari a 2^{16}, cioè 65536, in cui ciascuna locazione è considerata come contenente una quantità di 8 bit che rappresenta un valore di dati, un'istruzione o qualche altro elemento. Al giorno d'oggi, questo intervallo d'indirizzamento è considerato inadeguato per molte delle applicazioni che richiedono l'impiego di microprocessori a 16 bit o a 32 bit.

La Tab. 2.1 elenca le caratteristiche di microcomputer tipici con lunghezze di word di 8, 16 e 32 bit. Nella prima riga è mostrato l'intervallo decimale di un numero per la lunghezza di word di dati specificata. Poi segue il numero delle linee di segnali d'indirizzo ed infine è indicata la lunghezza della memoria. Quest'ultima è spesso specificata come un multiplo di 1024 (2^{10}) locazioni, che viene denominato "1K" di memoria. Quindi 16 linee d'indirizzo consentirebbero 65536 locazioni, cioè 64K di memoria.

Tab. 2.1 *Caratteristiche di microprocessori tipici.*

Caratteristica	**Famiglia a 8 bit**	**Famiglia a 16 bit**	**Famiglia a 32 bit**
Intervallo decimale di dati	da 0 255	da 0 a 65535	da 0 a 4294967925
Numero di linee d'indirizzo	da 14 a 16	da 16 a 24	32
Lunghezza tipica della memoria (1K = 1024 byte)	da 16K a 64K	da 64K a oltre 16 milioni (16384K)	Qualsiasi dimensione fino a 4194304K

Nota: la famiglia di 16 bit include l'MC68000 e processori simili.

Esempio 2-1

Una misura approssimativa della potenza di elaborazione di una CPU è determinata dal numero di linee di segnale del bus indirizzi per la larghezza del bus dati per la frequenza di clock in milioni di cicli a secondo (MHz). Quindi per l'MC68020 tale prodotto risulta:

32 x 32 x 16.7 = 17.100

usando la versione a 16.7 MHz. Per un confronto, l'MC68000 ha un prodotto di:

24 x 16 x 12.5 = 4800

per la versione a 12.5 MHz. Il rapporto indica un miglioramento di prestazioni di 3.56 volte a favore dell'MC68020. Questa non è una stima irragionevole, come si può verificare confrontando direttamente il tempo richiesto per eseguire vari programmi in un sistema basato sull'MC68000 col tempo richiesto da un computer che abbia un MC68020 come processore centrale.

Esempio 2-2

Come esempio di utilizzazione del bus interno di sistema, si supponga che un processore con 32 linee d'indirizzo e 32 linee di dati debba eseguire un'istruzione. L'istruzione è lunga 32 bit. Essa è memorizzata in due word di 16 bit alle locazioni 10 e 12 della memoria, come mostrato nella Tab. 2.2. L'ipotetica istruzione scritta in forma mnemonica è CLR 100. Questa istruzione azzera il contenuto della locazione numero 100 della memoria; lo zero è rappresentato come una stringa di 16 bit {0} nella locazione 100. La prima word di 16 bit dell'istruzione contiene l'operazione ed indica che l'indirizzo della locazione interessata è posto nella word successiva. L'indirizzo 100 è memorizzato in binario nella locazione 12.

Tab. 2.2 *Contenuto della memoria per l'istruzione CLR 100.*

Indirizzo di memoria (decimale)	Contenuto della memoria (binario)
10	0100 0010 0111 1000
12	0000 0000 0110 0100
14	(istruzione successiva)
.	.
.	.
.	.

Nota: le locazioni pari contengono una word di 16 bit che è considerata come due locazioni di byte da 8 bit ciascuna.

La Fig. 2.7 illustra lo stato delle linee di segnale al crescere del tempo, mentre il processore esegue l'istruzione. Il prelievo dell'istruzione richiede un ciclo di lettura da parte della CPU, per determinare l'operazione e la locazione da azzerare. Per ogni operazione di lettura, la memoria risponde presentando il contenuto della locazione indirizzata sulle linee di dati. C'è un lieve ritardo mentre le linee dei segnali di dati cambiano per assumere i nuovi valori. Dopodiché, la CPU decodifica l'istruzione CLR e la esegue, scrivendo gli zeri nella locazione 100 della memoria, tramite le linee di segnali di dati, dopo aver fornito i segnali di controllo per la scrittura. Poi sarà prelevata ed eseguita l'istruzione successiva.

Linee di segnale

Indirizzo — 0000 0000 0000 0000 0000 0000 0000 1010 — 0000 0000 0000 0000 0000 0000 0110 0100

Dati — 0100 0010 0111 1000 0000 0000 0110 0100 — 0000 0000 0000 0000 0000 0000 0000 0000*

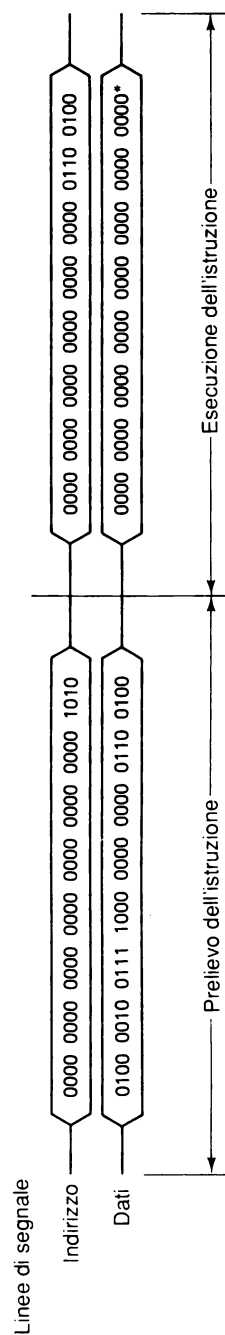
Prelievo dell'istruzione — Esecuzione dell'istruzione

**Nota:* soltanto 16 bit sono scritti nella locazione 100.

Fig. 2.7 *Stati delle linee di segnale al crescere del tempo. (Esempio 2.2)*

In questo esempio, alle locazioni di memoria di 16 bit sono stati assegnati indirizzi pari, anziché valori consecutivi, poiché i processori come l'MC68020 possono indirizzare operandi di 8 bit nella memoria. La locazione di word 10 contiene due byte alle locazioni 10 e 11. Ciascuna istruzione consiste di due o più word di 16 bit.

La sequenza di Fig. 2.7 è tipica di un processore che esegue una semplice istruzione come CLR. Sebbene l'operazione sia funzionalmente simile a quella illustrata, l'esecuzione di questa istruzione da parte dell'MC68020 sarebbe in realtà più complicata. Questo argomento sarà ripreso nel cap. 4, a proposito delle caratteristiche di prelievo di un'istruzione dell'MC68020.

ESERCIZI

2.2.1 L'MC68020 può indirizzare operandi di byte (8 bit), di word (16 bit) o di longword (32 bit). Per ciascuna lunghezza di operando, si tracci un diagramma che illustra l'organizzazione di un buffer di otto byte nella memoria a partire dalla locazione 1000. Si numerino le posizioni di bit e gli indirizzi. Un buffer è un'area di memoria che contiene temporaneamente i dati durante i trasferimenti di I/O.

2.2.2 Quanti bit (linee d'indirizzo) sono necessari per indirizzare una memoria con:
(a) 4096 locazioni?
(b) 65536 locazioni?
(c) 16777216 locazioni?

2.2.3 Qual è la massima dimensione di un programma per un microprocessore con:
(a) 16 linee d'indirizzamento?
(b) 20 linee d'indirizzamento?
(c) 24 linee d'indirizzamento?
(d) 32 linee d'indirizzamento?

2.2.4 L'MC68020 ha 32 linee d'indirizzamento e può indirizzare ogni byte di 8 bit nella memoria. Quante linee d'indirizzamento sarebbero necessarie se l'MC68020 potesse indirizzare soltanto word di 8 bit?

2.2.5 Basandosi sugli articoli pubblicati nelle riviste di computer, si determinino le più importanti applicazioni dei microprocessori a 8, 16 e 32 bit.

2.2.6 Si discutano alcune limitazioni dei microprocessori a 8 bit che hanno tipicamente circa 40 linee di segnale. Si considerino le restrizioni sull'insieme di istruzioni e le funzioni d'interfacciamento che possono essere eseguite dalla CPU come pure la sua velocità operativa. Si forniscano esempi di microprocessori a 8 bit che superano alcune delle limitazioni tramite ripartizione (multiplexing) nel tempo di linee di segnale selezionate.

2.3 TRE PUNTI DI VISTA SUL MICROPROCESSORE

I moderni microprocessori come l'MC68020 sono incorporati nei sistemi di computer per controllare l'attività globale del sistema. Questi processori sono in grado di dirigere l'attività del sistema eseguendo programmi e svolgendo le necessarie funzioni d'ingresso/uscita. Per di più, i moderni processori distinguono tra gli stati o modalità di supervisore e di utente per l'elaborazione, consentono la gestione e la protezione della memoria e rivelano vari tipi di errori.

Le caratteristiche di programmazione dei processori a 32 bit includono un insieme di istruzioni generale e potente, come pure un certo numero di modi distinti per fare riferimento ad un operando nella memoria (modalità d'indirizzamento). Molti di questi processori offrono la capacità di supporto a tecniche speciali di programmazione ed ausilii per il debugging.

L'interazione tra il processore e gli altri elementi hardware del sistema avviene tramite il bus di sistema, su cui "viaggiano" i segnali di controllo, gli indirizzi e i dati. La capacità e la flessibilità dei processori in questo senso è determinata dalle funzioni delle linee di segnali provenienti dal processore. Per esempio, le capacità avanzate di I/O e di interruzione alleviano l'esigenza di disporre di una grande quantità di hardware speciale in un sistema complesso.[3] La maggior parte dei processori a 32 bit forniscono lo stato del processore ed altre informazioni pertinenti ai circuiti esterni durante l'attività del processore. Questa è una caratteristica che semplifica il progetto dell'hardware, come sarà discusso nel cap. 13.

In questo paragrafo sono illustrate brevemente quelle caratteristiche dei processori a 32 bit che servono a soddisfare le esigenze di progettazione, programmazione ed interfacciamento di sistemi complessi. Sebbene il materiale presentato qui sia alquanto generale, esso forma la base per la comprensione di molte delle caratteristiche della famiglia dell'MC68020. La Fig. 2.8 riassume tre diversi punti di vista di un microprocessore. Il progettista di sistema, il programmatore in linguaggio assembler ed il progettista d'interfaccia sono interessati ognuno ad aspetti distinti del processore, cosicché i rispettivi punti di vista possono coincidere oppure divergere in qualche caso. Ovviamente il programmatore in linguaggio assembler ed il progettista d'interfaccia hanno i medesimi obiettivi nel collaudo di un prototipo di sistema, ma i rispettivi approcci nella realizzazione di un prodotto funzionante possono differire considerevolmente.

2.3.1 Progettazione del sistema

Il progettista di sistema è interessato all'attività globale del sistema, tra cui le prestazioni e l'affidabilità. Il progettista determina anche l'utilizzazione della memoria ed il modo in cui le aree di memoria occupate dal sistema operativo saranno

[3] In queste discussioni, un'interruzione è considerata come un segnale originato all'esterno del microcomputer che causa un trasferimento di controllo da un programma in esecuzione ad un programma speciale, al fine di eseguire l'elaborazione richiesta in risposta all'interruzione. Dopo il completamento del programma d'interruzione, il controllo torna al programma che era stato interrotto.

ORGANIZZAZIONE
CARATTERISTICHE DEL SISTEMA
VELOCITA' OPERATIVA
PROTEZIONE DEL SISTEMA E RIVELAZIONE DELL'ERRORE
COPROCESSORE O MULTIPROCESSORE
PROGRAMMAZIONE DEL SISTEMA
MULTIPROGRAMMAZIONE
MULTIUTENZA
UTILIZZAZIONE DELLA MEMORIA
PROTEZIONE DELLA MEMORIA E SEGMENTAZIONE
MEMORIA VIRTUALE
CAPACITA' D'INTERRUZIONE E DI I/O
NUMERO DI UNITA' PERIFERICHE
METODO DI PRIORITA' D'INTERRUZIONE

(a) *Progettista del sistema.*

ARCHITETTURA
PROGRAMMAZIONE
INSIEME DI ISTRUZIONI E TIPI DI DATI
RIFERIMENTO ALLA MEMORIA (MODALITA' D'INDIRIZZAMENTO)
TECNICHE SPECIALI
SUPPORTO DI MODERNE TECNICHE DI PROGRAMMAZIONE
ISTRUZIONI DI COPROCESSORE
DEBUGGING
RIVELAZIONE DELL'ERRORE
INFORMAZIONI DIAGNOSTICHE

(b) *Programmatore in linguaggio assembler.*

IMPLEMENTAZIONE
CARATTERISTICHE D'INTERFACCIAMENTO
FUNZIONI DELLE LINEE DI SEGNALE
STRUTTURA DI I/O E DI INTERRUZIONE
CONTROLLO DELL'ERRORE E CAPACITA' DI DEBUGGING
RIVELAZIONE DELL'ERRORE
INFORMAZIONI DIAGNOSTICHE

(c) Progettista di interfaccia

Fig. 2.8 *Tre punti di vista del processore.*

protette se s'impiega una memoria di lettura/scrittura. Questa protezione è essenziale se il sistema è adoperato per lo sviluppo di software, in cui possono presentarsi errori d'indirizzamento e programmi che superano i limiti previsti. Quando molte unità periferiche sono collegate al sistema, il progetto della porzione di I/O del sistema è d'importanza critica nel garantire un coordinamento appropriato tra i programmi e l'hardware durante i trasferimenti di dati. La Tab. 2.3 riassume gli aspetti di progettazione del sistema discussi in questo paragrafo.

Tab. 2.3 *Considerazioni di sistema.*

Caratteristica	Funzione
SISTEMA	
Velocità operativa	Una misura delle prestazioni del sistema
Protezione del sistema e rivelazione degli errori	Distinguere il modo di supervisore dal modo di utente
Coprocessore o multiprocessore	Migliorare le prestazioni del sistema
PROGRAMMAZIONE DEL SISTEMA	
Multiprogrammazione o multiutenza	Condividere l'uso della CPU tra utenti o programmi
UTILIZZAZIONE DELLA MEMORIA	
Protezione della memoria e segmentazione	Impedire ai programmi di utente d'interferire con la memoria allocata al sistema operativo o con altri programmi di utente
Memoria virtuale	Possibilità d'indipendenza dei programmi dalla memoria fisica
CAPACITA' DI I/O E D'INTERRUZIONE	
Numero di unità periferiche ammesse e metodo di priorità d'interruzione	Determinare la capacità di I/O del sistema (cioè, il numero di dispositivi ed i tempi di risposta per il trasferimento di dati)

Nota: I programmi di utente sono programmi eseguiti nel modo operativo di *utente* del processore.

Velocità operativa. Un criterio importante adottato per misurare le prestazioni di un microcomputer è la velocità operativa con cui esegue un certo programma. Un elemento fondamentale — benché non l'unico — che determina la velocità operativa è il massimo numero di cicli di clock al secondo che la CPU può raggiungere in fase di esecuzione. Come discusso nel par. 1.2, l'MC68020 è prodotto in versioni che offrono un'ampia possibilità di selezione della velocità operativa. Per esempio, l'MC68020RC16 può eseguire 16.7 milioni di cicli di clock al secondo, mentre l'MC68020RC12 ne esegue soltanto 12.5 milioni. Queste velocità sono utili nel confronto delle prestazioni di sistemi diversi basati su differenti versioni del processore. Tuttavia, il numero di cicli al secondo della CPU non sempre è in una relazione semplice col numero di istruzioni al secondo che possono essere eseguite, talvolta misurate in "MIPS" (milioni di istruzioni al secondo). Ciò è dovuto al fatto che istruzioni differenti possono richiedere numeri differenti di cicli di clock per l'esecuzione.

Oltre alla difficoltà di porre in relazione il numero di cicli di clock al secondo della CPU con la velocità di esecuzione delle istruzioni, non è semplice prevedere il tempo richiesto da un computer basato sull'MC68020 per eseguire un segmento di programma. Il motivo di ciò è la presenza della memoria cache sul chip della CPU e la capacità della CPU di "sovrapporre" l'esecuzione di più istruzioni alla volta in qualche caso. Altri fattori che influiscono sul tempo complessivo richiesto per l'esecuzione di un programma sono la struttura del bus e le caratteristiche della memoria. Alcune stime di temporizzazione specifiche dell'MC68020 saranno fornite nel par. 13.3.

Protezione del sistema e rivelazione degli errori. Per impedire che gli errori nei programmi applicativi influiscano sul funzionamento del sistema nel suo complesso (o l'uno con l'altro), il processore MC68020 dispone di due modalità di esecuzione: il modo di *supervisore* e il modo di *utente*. I programmi eseguiti nel modo di supervisore hanno il controllo completo delle funzioni del processore e del sistema. Ovviamente le routine importanti del sistema operativo sono eseguite nel modo di supervisore. Tra queste sono comprese tutte le routine per gestire l'I/O e le interruzioni.[4] Di solito, i programmi applicativi vengono eseguiti nel modo di utente. In questa modalità, certe istruzioni del processore ed eventualmente certe aree di memoria sono inaccessibili al programma applicativo. La selezione della modalità per vari programmi è determinata dal progettista del sistema e le transizioni tra le due modalità sono accuratamente controllate.

La Fig. 2.9 illustra un esempio di operazioni del sistema in funzione del tempo in cui il controllo viene passato ad un programma di utente e restituito al sistema operativo. Il ritorno al modo di supervisore può essere causato dal completamento del programma, da un errore rivelato dalla CPU o da un'interruzione.

I processori quali l'MC68020 consentono di rivelare ed *intrappolare* il manifestarsi di certi errori durante l'esecuzione del programma in entrambe le modalità.

[4] Una routine è di solito un breve segmento di programma che ha lo scopo di eseguire una funzione specifica (come il trasferimento di un valore di dati o una funzione simile).

Fig. 2.9
Stati del processore.

Tempo di esecuzione

Modalità di supervisore

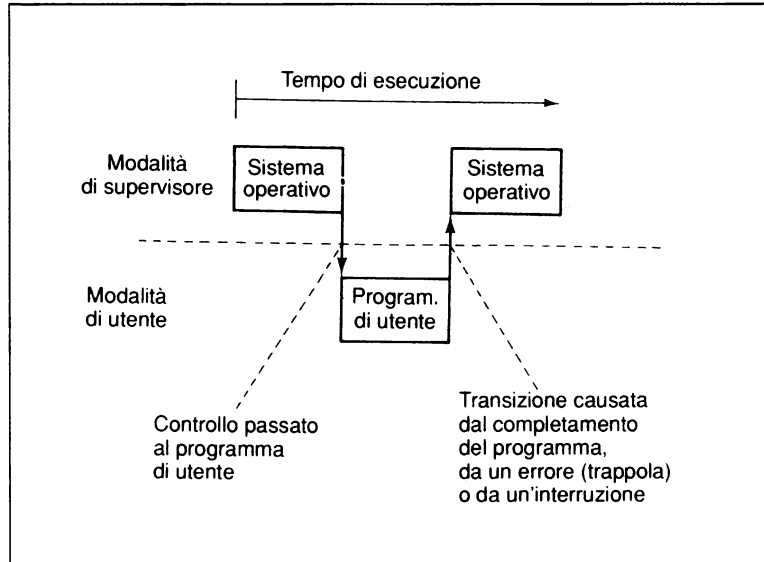
Sistema operativo

Sistema operativo

Modalità di utente

Program. di utente

Controllo passato al programma di utente

Transizione causata dal completamento del programma, da un errore (trappola) o da un'interruzione

Il meccanismo della trappola trasferisce il controllo del processore dal programma che ha fatto scattare la trappola ad una routine del software di sistema che gestisce la condizione di errore. L'esecuzione di un'istruzione non ammessa ("illegale") è un esempio di istruzione che fa scattare una trappola.[5] Nell'MC68020, anche un tentativo di divisione per zero in un'istruzione aritmetica causerà una trappola.

Certi errori dell'hardware possono essere rivelati usando il meccanismo dell'interruzione. Per esempio, il sistema potrebbe essere progettato per elaborare un'interruzione causata da una mancanza di alimentazione elettrica. In tal caso, la routine d'interruzione ordinerebbe alla CPU di salvare immediatamente tutte le informazioni che consentiranno, nel momento in cui l'alimentazione sarà ripristinata, la ripresa del programma che era stato interrotto.

Le interruzioni possono presentarsi in qualsiasi istante, in modo asincrono con l'esecuzione del programma. Le trappole, d'altro canto, si presentano esclusivamente come risultato dell'esecuzione di istruzioni del programma. Ogni tipo di trappola e d'interruzione possibile nell'MC68020 sarà descritto in dettaglio nel cap. 11.

Coprocessore o multiprocessore. Molte applicazioni richiedono operazioni speciali che di solito non fanno parte dell'insieme di istruzioni di un microprocessore di finalità generale. Per esempio, le applicazioni scientifiche e tecniche richiedono solitamente operazioni aritmetiche in virgola mobile e calcoli trigonometrici. Il coprocessore MC68881 con il suo insieme di istruzioni in virgola mobile

[5] Un'istruzione illegale è un'istruzione di linguaggio-macchina con una configurazione di bit non riconosciuta dalla CPU.

potrebbe essere incluso in un sistema per tali applicazioni specifiche. Un coprocessore fa aumentare notevolmente la velocità di esecuzione del sistema grazie all'opportunità di eseguire istruzioni speciali, non disponibili in un sistema che utilizza un programma per svolgere la medesima funzione, usando solamente le istruzioni della CPU per calcolare i valori. Un progetto alternativo senza coprocessore potrebbe includere dell'hardware speciale per svolgere tali funzioni.

Un coprocessore dell'MC68020 potrebbe essere un prodotto della Motorola oppure un'unità appositamente progettata, fornita da un'altra società. L'MC68020 dispone di istruzioni per il controllo del coprocessore, come pure dell'interfaccia di linee di segnale da collegare al coprocessore. Fino ad otto coprocessori separati sono ammessi in un sistema basato sull'MC68020.

Alcune applicazioni possono richiedere la condivisione del medesimo bus da parte di più microprocessori. In un siffatto computer *multiprocessore*, ciò può essere ottenuto accrescendo la velocità di esecuzione del sistema. Il computer della Masscomp, presentato nel cap. 1, adotta questo metodo. I sistemi multiprocessore possono essere usati per fornire un'affidabilità maggiore di quella possibile in un sistema con un singolo processore, se un processore può eseguire il programma di un altro che presenta qualche malfunzionamento. La Fig. 2.10 illustra una possibile configurazione di un sistema multiprocessore. I componenti del sistema possono essere interconnessi su un'unica piastra circuitale o possono rappresentare moduli singoli che comunicano tramite un bus di sistema quale il VMEbus descritto nel cap. 14.

Nota: L'unità di arbitrato di bus riceve le richieste dagli altri elementi del sistema e concede ad essi l'accesso al bus in base ad una priorità prestabilita. Ciò viene effettuato mediante le linee di segnale della porzione di controllo del bus.

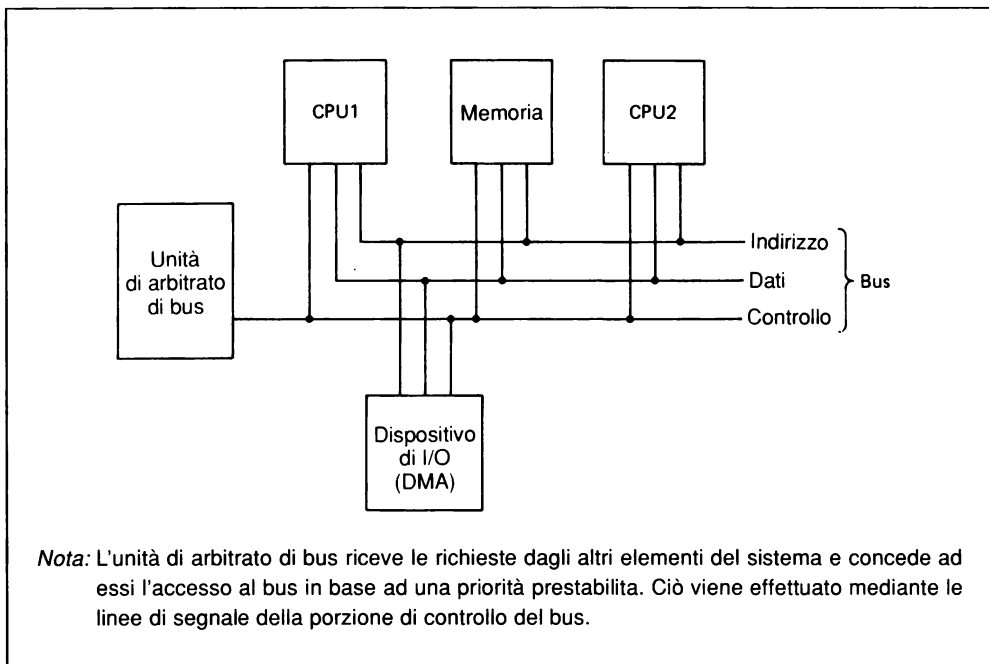



Fig. 2.10 *Diagramma semplificato di un sistema multiprocessore.*

L'MC68020 permette di realizzare sistemi in multielaborazione, in cui il medesimo bus è condiviso da diverse CPU o dispositivi d'ingresso/uscita che impiegano l'accesso diretto alla memoria (*Direct Memory Access:* DMA) per il trasferimento dei dati. In questo metodo, una sola unità, designata come "master" (padrone) del bus, è scelta per utilizzare il bus in un certo istante. L'MC68020 semplifica il progetto della circuiteria di arbitrato del bus poiché dispone delle linee di segnali di controllo per rilasciare o assumere il controllo del bus a seconda delle necessità. Come parte del supporto alla famiglia dell'MC68020, la Motorola produce un "modulo di arbitrato di bus", cioè un circuito integrato che esegue la funzione di arbitrato del bus, come illustrato in Fig. 2.10. Certe istruzioni dell'MC68020, come TAS (*Test and Set Operand:* esame e assegnazione dell'operando) permettono che più processori condividano un'area di memoria comune in un'applicazione multiprocessore, senza il rischio di un accesso simultaneo alla medesima locazione di memoria. Se una delle CPU nel sistema esegue questa istruzione, allora nessun altro componente del sistema potrà assumere il controllo del bus finché l'istruzione non avrà completato le sue operazioni di lettura del contenuto di una locazione di memoria, esame o modifica dell'operando, e infine scrittura del risultato ancora nella memoria. L'istruzione TAS sarà descritta nel cap. 8.3. Altre caratteristiche dell'MC68020 che offrono un supporto ad un sistema di computer con più processori saranno discusse nel par. 12.6.

Programmazione di sistema: computer di multiprogrammazione e multiutenza. L'MC68020 dispone di caratteristiche atte a gestire la multiprogrammazione. Un sistema operativo di *multiprogrammazione* permette l'esecuzione concorrente (simultanea) di più programmi indipendenti. Talvolta si designa come "task" ogni unità di programma completa. Ad un osservatore l'elaborazione dei task appare come se le loro esecuzioni si sovrapponessero nel tempo. In questi sistemi, diversi programmi o task possono trovarsi in vari stadi di esecuzione in un certo istante. Un task può prevalere su un altro ed assumere il controllo della CPU, nel rispetto delle priorità dei task assegnate dal sistema operativo. In tal caso, tutte le informazioni pertinenti al task che ha dovuto cedere la precedenza devono essere salvate nella memoria, in modo che l'esecuzione di quel task possa essere ripresa quando il controllo sarà restituito ad esso. Il concetto è simile a quello della transizione di stato del processore illustrata nella Fig. 2.9, anche se di solito i task vengono eseguiti nel modo di utente dell'MC68020. Il salvataggio dell'informazione ed il passaggio del controllo al nuovo task è noto come *commutazione di contesto.* L'MC68020 facilita in vari modi tale commutazione di contesto, come si vedrà dalla discussione nel par. 12.5.

Quando il computer è utilizzato per elaborare più task in modo concorrente, viene impiegato un sistema operativo che consente la multiprogrammazione. I task possono essere in relazione reciproca come parti di un'unità di programma più grande, oppure ciascun task può essere indipendente da tutti gli altri. Il sistema operativo VERSAdos della Motorola, presentato nel cap. 1, è definito sistema operativo *multitasking* (multiprogrammazione). Esso è anche un sistema operativo multiutente. In quest'applicazione, un certo numero di programmatori — denominati "utenti" — può condividere simultaneamente il sistema del computer. Ciascun programmatore ha l'impressione di avere l'uso esclusivo della macchina, a causa

del modo in cui il sistema operativo commuta tra i vari task mentre il computer esegue i programmi.

Per gli scopi di questo libro, le caratteristiche dei vari tipi di sistemi operativi non sono d'importanza primaria. Il punto saliente è che l'MC68020 offre al progettista di sistema una grande flessibilità quando si tratta di progettare o selezionare un sistema operativo. La separazione del modo di utente da quello di supervisore e la capacità di commutazione di contesto sono caratteristiche del microprocessore importanti per il programmatore che crea il software di sistema. Inoltre, le aree di memoria selezionate per essere usate da un certo task possono essere protette contro l'accesso da parte di un altro task, come sarà discusso nel prossimo paragrafo.

Utilizzazione della memoria: protezione e segmentazione. Poiché l'MC68020 ha la capacità d'indirizzare 2^{32} locazioni di byte, pochissimi sistemi sono limitati da una mancanza di spazio di memoria. Nella maggioranza dei sistemi, sia il sistema operativo che i programmi applicativi possono essere contenuti nella memoria senza conflitti. Il progettista di sistema determina l'allocazione della memoria specificando il numero richiesto di locazioni per ciascun programma. Specialmente per quei sistemi utilizzati per lo sviluppo di programmi o per applicazioni multiutente, dev'essere disponibile un mezzo che impedisca ad un programma in esecuzione di accedere (leggendo o scrivendo) a qualsiasi locazione di memoria non assegnata a tale programma. Questa protezione è di solito fornita allo spazio di memoria del sistema operativo per impedire l'accesso da parte di programmi in esecuzione nel modo di utente.

Nei sistemi basati sull'MC68020, la memoria può essere protetta in questo modo mediante appositi circuiti di gestione della memoria, come la *Memory Management Unit* (MC68851) discussa nel par. 1.2. L'MC68020 indica il tipo di accesso come supervisore o utente per mezzo di tre sue linee di segnali di controllo. Simultaneamente, i circuiti di gestione della memoria confrontano la locazione di memoria indirizzata con l'intervallo valido per il modo assegnato al programma. Una violazione viene indicata alla CPU dalla circuiteria di gestione della memoria. Quindi, se la CPU fornisce il modo e l'indirizzo per ciascun accesso alla memoria come fa l'MC68020, allora la protezione delle aree di memoria è facilmente ottenibile. Nessuno dei precedenti microprocessori della classe di 8 bit disponeva di questa caratteristica.

Memoria virtuale. Il processore MC68020 e la *Memory Management Unit* possono essere combinati per fornire il supporto hardware sia per la protezione della memoria che per la memoria virtuale. Un sistema di memoria virtuale ha il vantaggio di permettere l'impiego di programmi di qualsiasi dimensione anche se lo spazio di memoria fisica è relativamente piccolo, almeno in teoria. Ulteriori discussioni dei sistemi con memoria virtuale con l'MC68020 sono reperibili nei riferimenti bibliografici per questo capitolo riportati nell'appendice E. Questo argomento sarà discusso nel par. 12.4.

Capacità di I/O e d'interruzione. Il progettista di sistema specificherà il numero e il tipo di dispositivi periferici necessari per soddisfare i requisiti di un'applicazione. Il progetto successivo diviene complicato quando devono essere collegati molti dispositivi, poiché unità diverse hanno diversi requisiti di tempo per completare i trasferimenti di dati. Per esempio, una stampante parallela impiega molto più tempo per stampare i caratteri di quanto non ne richieda un'unità a disco per memorizzarli. A causa di queste differenze nei tempi, il sistema viene progettato in modo che ciascun dispositivo possa inviare una *richiesta d'interruzione* attraverso parecchie linee di segnali di controllo che vanno dall'interfaccia alla CPU. Una richiesta viene inviata quando il dispositivo è pronto a ricevere o a trasmettere i dati o quando viene rivelata una condizione di errore. Dal punto di vista di un programma in esecuzione, l'interruzione causa una sospensione fino a quando la routine d'interruzione non avrà completato il trasferimento dei dati o gli altri tipi di elaborazione richiesti. Una routine di questo tipo viene eseguita nel modo di supervisore in un sistema basato sull'MC68020.

Il meccanismo d'interruzione è un fattore determinante della capacità di I/O di un sistema quando un certo numero di dispositivi periferici sono collegati al microcomputer. In processori quali l'MC68020, la circuiteria d'interruzione fa parte della CPU. Fisicamente, ciò significa che più linee di controllo (tre per l'MC68020) sono dedicate alle richieste d'interruzione da dispositivi esterni. Alle otto (2^3) possibili richieste d'interruzione sono assegnate priorità tali che un'interruzione di livello superiore interromperà l'esecuzione di una routine avviata da una richiesta d'interruzione di livello inferiore. In teoria, fino a otto routine potrebbero essere contemporaneamente in vari stati di esecuzione in un sistema basato sull'MC68020.

Per ogni livello d'interruzione, ad un certo numero di dispositivi potrebbe essere stata assegnata la medesima priorità. In tal caso, sarebbe richiesta una circuiteria esterna per risolvere i conflitti, se due o più dispositivi richiedessero contemporaneamente un'interruzione al medesimo livello o se più interruzioni fossero in sospeso (cioè, in attesa del completamento della routine d'interruzione a quel livello). La CPU della Motorola è in grado di gestire fino a 192 dispositivi distribuiti secondo i requisiti del sistema attraverso i suoi otto livelli d'interruzione. Tuttavia, una configurazione siffatta richiederebbe una notevole progettazione di hardware per controllare i dispositivi che potrebbero interrompere la CPU al medesimo livello d'interruzione. La struttura dell'interruzione sarà spiegata in maggiori dettagli nei capp. 11 e 13.

2.3.2 Programmazione in linguaggio assembler

La facilità con cui un programma può essere creato, corretto e provato per soddisfare un'applicazione specifica dipende in gran parte dalle caratteristiche del processore, anziché dal software di sviluppo, se il programma è scritto in linguaggio assembler. Editor, assemblatori ed altri ausilii allo sviluppo sono disponibili in vari gradi di qualità e di efficienza, ma un buon sistema di sviluppo non può sopperire all'inadeguatezza di un processore. I processori della classe dell'MC68020 sono adeguati per la maggioranza delle applicazioni, grazie ai loro potenti insiemi di

istruzioni, alle numerose modalità d'indirizzamento e ad altre caratteristiche speciali che non erano disponibili nei microprocessori precedenti. La Tab. 2.4 riassume alcune caratteristiche di un microprocessore che servono a determinare la sua capacità nel soddisfare i requisiti di programmazione di software avanzato.[6]

Tab. 2.4 *Considerazioni di programmazione.*

Caratteristica	Scopo
PROGRAMMAZIONE	
Insieme di istruzioni e tipi di dati	Determina l'efficienza e la facilità di programmazione.
Modalità d'indirizzamento	Indica la capacità di consentire la creazione di strutture di dati nella memoria.
Tecniche speciali	Offre l'opportunità di creare programmi e sistemi avanzati.
DEBUGGING	
Rivelazione degli errori e informazioni diagnostiche	Utile per individuare certi errori e determinarne la causa.

Insieme di istruzioni e tipi di dati. L'*insieme di istruzioni* di un microprocessore è l'insieme di tutte le istruzioni in linguaggio-macchina disponibili al programmatore. Ciascuna istruzione può essere descritta dalla sua operazione o funzione e dal numero e tipo di operandi su cui agisce. Per esempio, l'istruzione di addizione binaria dell'MC68020 somma due interi con segno. L'MC68020 consente anche la sottrazione, la moltiplicazione e la divisione di due interi siffatti. Inoltre, sono disponibili le istruzioni per sommare e sottrarre numeri decimali.[7] Quindi l'MC68020 ha un insieme di istruzioni piuttosto completo per operare su operandi numerici.

[6] L'espressione *architettura di computer* è talvolta impiegato per denotare l'intero insieme di caratteristiche di un sistema di computer che sono d'importanza per il programmatore. Una descrizione dell'architettura dovrebbe includere una definizione dell'organizzazione globale del sistema come pure una discussione completa delle caratteristiche di programmazione della CPU. L'insieme di istruzioni del processore e le modalità d'indirizzamento costituiscono due delle caratteristiche più importanti della CPU per tale descrizione.

[7] I numeri decimali sono rappresentati nella memoria come sequenze di bit codificate in una rappresentazione nota come decimale codificata in binario (*Binary Code Decimal*: BCD). Nel cap. 3 saranno discussi i tipi di operandi ammessi con le istruzioni dell'MC68020.

Per contro, i precedenti microprocessori a 8 bit avevano un insieme di istruzioni più limitato per le operazioni aritmetiche. Ad esempio, le istruzioni di divisione e di moltiplicazione non erano disponibili. Per svolgere tali operazioni, si dovevano creare delle routine basate sulle operazioni di addizione e di sottrazione, cosicché la moltiplicazione veniva eseguita mediante una somma ripetuta. Nella maggior parte dei casi, le istruzioni equivalenti dei processori a 32 bit sono più potenti ed efficienti rispetto a quelle delle classi di 8 bit e di 16 bit. Ciò semplifica la programmazione ed aumenta la velocità di esecuzione di programmi equivalenti. Un esempio in merito è stato fornito nel par. 2.1, discutendo la lunghezza della word di dati. L'MC68020 può eseguire operazioni aritmetiche su operandi la cui lunghezza può essere considerata di 8, di 16 o di 32 bit. Quando i processori a 8 bit disponevano di istruzioni equivalenti, la lunghezza dell'operando era tipicamente di soli 8 bit.

La Fig. 2.11 mostra i tipi di dati disponibili con l'MC68020 ed il suo coprocessore MC68881. L'insieme di istruzioni combinato dispone di operazioni su numeri binari e su interi decimali e in virgola mobile. L'MC68020 ha anche un insieme di istruzioni di trattamento di bit che consentono operazioni su singoli bit di un operando di 8 o di 32 bit. Il bit selezionato può essere esaminato o posto a {1} o a {0} dalle istruzioni di bit. Tali operazioni sono importanti quando lo stato di un dispositivo è indicato o definito come valore binario (che rappresenta cioè uno stato ON o OFF). Quando dev'essere trattato un gruppo di bit, s'impiegano le istruzioni dell'MC68020.

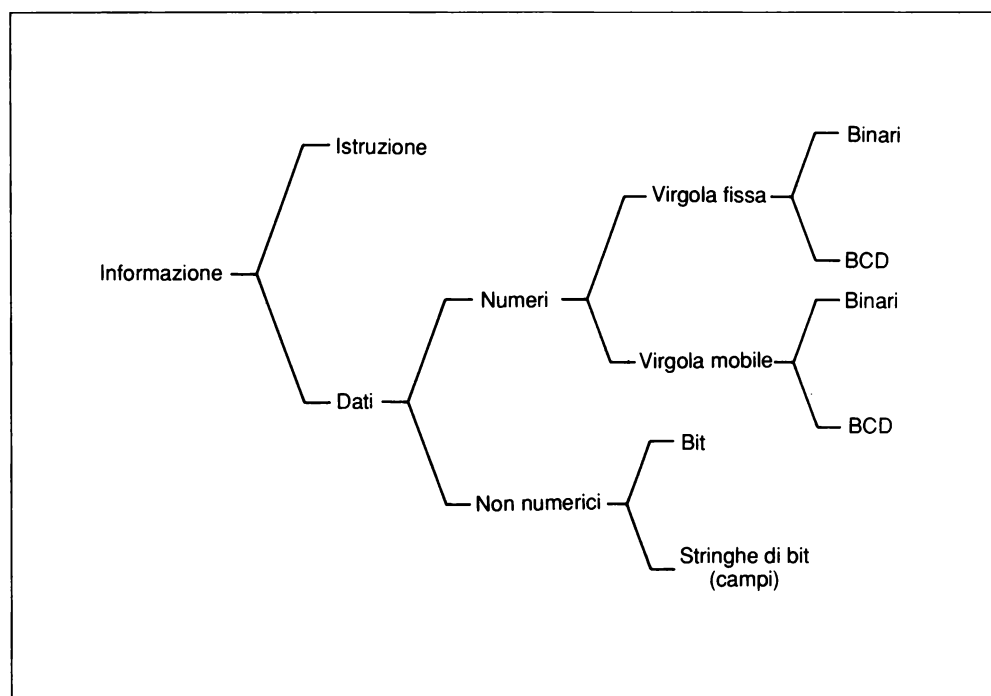



Fig. 2.11 *Tipi di dati.*

Le istruzioni del campo di bit consentono al programmatore di trattare sequenze di bit nella memoria denominate "campi". Un campo di bit può avere un numero variabile di bit. Il valore potrebbe rappresentare variabili in un linguaggio di compilatore come il C, oppure il campo potrebbe essere una "mappa di bit" di porzioni dello schermo video in un'applicazione grafica. Altre istruzioni sono disponibili per il trasferimento di dati e per operazioni logiche, di scorrimento e rotazione, e di controllo del programma (salti e chiamate di subroutine). Oltre a queste istruzioni per il trattamento dei dati e per effettuare i calcoli, l'MC68020 ha un certo numero di istruzioni di supporto a tecniche speciali di programmazione. L'insieme di istruzioni dell'MC68020 sarà presentato nel cap. 4 e discusso in dettaglio nei capitoli successivi.

Riferimento alla memoria: modalità d'indirizzamento. Quando si confrontano i processori, vengono esaminati il numero e il tipo di istruzioni (inclusi gli operandi ammessi) per quanto concerne le loro capacità e flessibilità. Se gli operandi sono mantenuti nella memoria, l'indirizzo di un operando può essere specificato in un'istruzione come un intero di 32 bit. Questo metodo d'indirizzare direttamente ciascun operando è di solito noto come *indirizzamento assoluto*. Sono possibili anche altri metodi per indirizzare gli operandi; essi vengono denominati *modalità o modi d'indirizzamento*. Per esempio, l'MC68020 ha 18 modi d'indirizzamento distinti. Le istruzioni dell'MC68020 devono specificare non solo l'operazione da eseguire, ma anche il modo d'indirizzamento impiegato per far riferimento a ciascun operando indirizzato dall'istruzione. La CPU calcola durante l'esecuzione un indirizzo assoluto — talvolta denominato *indirizzo effettivo* in questo contesto — per ogni operando. In termini del numero di modalità d'indirizzamento, il sistema dell'MC68020 è più simile a un grande computer che ad un microcomputer, poiché i precedenti microprocessori avevano di solito capacità d'indirizzamento limitate. Per esempio, una certa varietà di strutture di dati, come liste ed array, può essere creata facilmente nella memoria con le modalità d'indirizzamento dell'MC68020. Queste ed altre strutture di dati saranno descritte nel cap. 9.

Tecniche speciali. L'MC68020 dispone di un certo numero di istruzioni per facilitare la progettazione e la programmazione di sistemi operativi, compilatori e programmi applicativi avanzati. Alcune di queste istruzioni consentono la creazione di programmi strutturati o modulari; altre sono disponibili per consentire ai programmi nelle modalità di utente o di supervisore di controllare l'attività del sistema in vari modi. È disponibile anche un insieme completo di istruzioni per controllare un coprocessore. Queste istruzioni sono di finalità generale ed il loro scopo è quello di controllare un coprocessore progettato "ad hoc" che può essere incluso in un sistema basato sull'MC68020.

Le moderne tecniche di programmazione richiedono che i programmi siano modulari per facilitare il debugging e i test. Ciascun modulo esegue una funzione definita concisamente e la creazione di un programma completo avviene collegando i moduli tra loro.[8] L'MC68020 ha un certo numero di istruzioni di supporto alla

[8] Nei programmi FORTRAN, i moduli sono denominati *subroutine*. Parametri quali indirizzi o dati vengono passati tra i moduli durante l'esecuzione del programma.

programmazione modulare, tra cui le istruzioni che richiamano le subroutine (i moduli). In aggiunta, l'MC68020 permette che i parametri siano facilmente trasferiti tra i moduli per mezzo della sua istruzione LINK.[9] Questa istruzione combina diverse operazioni che normalmente richiederebbero un breve segmento di programma per essere eseguite su altri processori.

Un certo numero di istruzioni dell'MC68020 sono utili per controllare l'attività del sistema. Per esempio, quando un'istruzione TRAP viene eseguita in un programma in modo di utente, il controllo viene restituito al modo di supervisore. Ciò è utile quando si chiamano routine del sistema operativo da un programma di utente.

L'MC68020 ha inoltre un insieme di istruzioni di controllo del sistema riservate ai programmi eseguiti in modo di supervisore. Queste istruzioni controllano lo stato del sistema o i dispositivi hardware esterni. Per esempio, l'istruzione RESET serve per inizializzare i chip periferici durante la fase d'inizializzazione del sistema operativo. Tali istruzioni sono trattae nei capp. 10 e 11.

Istruzioni del coprocessore. L'MC68020 ha un gruppo di istruzioni di finalità generale che si riferiscono specificamente ad un coprocessore. Queste istruzioni consentono il trasferimento di comandi e di dati tra la CPU ed il coprocessore, ma l'interfaccia di linee di segnale e l'insieme di istruzioni fondamentale sono definite e fornite dall'MC68020. Quando un coprocessore progettato "ad hoc" viene aggiunto al sistema basato sull'MC68020, tali istruzioni sono impiegate per creare i programmi che controlleranno il coprocessore.

Quando un coprocessore è un dispositivo della Motorola come l'MC68851 o l'MC68881, il suo insieme di istruzioni è fissato ed appare come un'estensione dell'insieme di istruzioni dell'MC68020. In un sistema con un MC68881, per esempio, l'istruzione FADD del linguaggio assembler indica l'addizione in virgola mobile che utilizza il coprocessore, mentre il codice mnemonico ADD designa l'addizione di interi destinata ad essere eseguita dall'MC68020. Le istruzioni del coprocessore saranno discusse nel cap. 12.

Questi esempi delle capacità dell'MC68020 come processore programmabile non danno che una minima idea della sua potenza e flessibilità. Molti dei prossimi capitoli sono dedicati ad esplorare più dettagliatamente l'insieme di istruzioni dell'MC68020 e i relativi concetti.

Debugging. L'MC68020 possiede varie caratteristiche che servono nel debugging e nel test di programmi. Il meccanismo per rivelare un errore è una trappola che "scatta" allorché viene eseguita un'istruzione che ha causato un errore. Per esempio, un'istruzione non ammessa o un tentativo di divisione per zero fa scattare una trappola. Anche certi errori d'indirizzamento e determinate condizioni aritmetiche sono causa di trappole. Quando scatta una trappola, il controllo viene

[9] LINK è il codice mnemonico in linguaggio assembler dell'istruzione, come si vedrà nel cap. 9.

passato ad una specifica routine del sistema operativo che esegue qualsiasi elaborazione richiesta per il tipo di errore rivelato. Le informazioni sullo stato del processore e sull'indirizzo dell'istruzione responsabile vengono salvate nel momento in seguito al verificarsi di una trappola, per facilitare la diagnosi del problema.

Il processore MC68020 ha due modalità operative di "traccia". In una di esse, la CPU esegue soltanto un'istruzione e poi entra in una routine di eccezione che può essere programmata per facilitare il debugging. L'altra modalità di traccia consente alla CPU di eseguire un programma finché non viene incontrata un'istruzione che modifica la sequenza di esecuzione. La seconda funzione di traccia è denominata "traccia sul cambio di flusso (del programma)". Ad esempio, un cambio di flusso avviene quando si raggiunge un'istruzione di salto o una trappola. Servendosi di questa opportunità, il programmatore può ignorare il flusso sequenziale di un programma ai fini del debugging, concentrandosi solamente sui punti di decisione del programma stesso. Anziché proseguire l'esecuzione nel nuovo segmento di programma, il controllo viene trasferito alla routine di eccezione associata con questa modalità di traccia allorché si verifica il cambio di flusso. Entrambe le modalità di traccia usano una trappola per restituire il controllo alla routine di debugging. La funzione di traccia viene attivata dal sistema operativo o dal monitor su richiesta del programmatore. Le modalità di traccia saranno discusse in dettaglio nel par. 11.3.

2.3.3 Progettazione dell'interfaccia

Il progettista d'interfaccia è interessato all'*implementazione* del sistema di computer quando progetta e collauda le interfacce. Le caratteristiche elettriche e funzionali delle linee di segnale del processore determinano il progetto della circuiteria che connette il processore ai dispositivi esterni. Gli aspetti funzionali delle linee di segnale determinano la loro funzione. Le caratteristiche elettriche comprendono le proprietà di temporizzazione dei segnali, i livelli di tensione ed altri dettagli importanti per i progettisti dei circuiti. Quando in questo libro si discute il progetto dell'interfaccia, viene posto in evidenza l'aspetto funzionale anziché i dettagli elettrici. Questo sottoparagrafo considera le caratteristiche d'interfacciamento e quelle di debugging dell'hardware dell'MC68020, elencate nella Tab. 2.5. Il cap. 13 è dedicato ad una discussione dettagliata delle caratteristiche d'interfacciamento dell'MC68020.

Linee di segnale dell'MC68020. La Fig. 2.12 mostra un diagramma semplificato dell'MC68020 che illustra le principali classi di linee di segnale connesse al bus di sistema. Queste linee sono fisicamente connesse al processore tramite i 114 piedini del contenitore di circuito integrato.[10] Trentadue linee di segnale sono usate per indirizzare una word di memoria, mentre 2 delle 11 linee di controllo del

---

{[10] La configurazione fisica dell'MC68020 è descritta in dettaglio nel cap. 4. Il contenitore in questione ha dimensioni approssimative di 1.35 pollici (34 mm) x 1.35 pollici ed è dotato di piedini lunghi circa 0.2 pollici (5 mm).

Tab. 2.5 *Considerazioni d'interfacciamento.*

Caratteristica	Impiego
INTERFACCIAMENTO	
Funzione delle linee di segnale	Indica lo stato o l'operazione richiesta dalla CPU o da dispositivi esterni. Consente il trasferimento di dati tra la CPU e i dispositivi esterni.
Struttura d'interruzione e di I/O	Consente ai dispositivi esterni di comunicare con la CPU in modo regolare.
CONTROLLO DELL'ERRORE E DEBUGGING	
Rivelazione degli errori hardware e informazioni diagnostiche	Assiste nel debugging o nel rimedio agli errori.

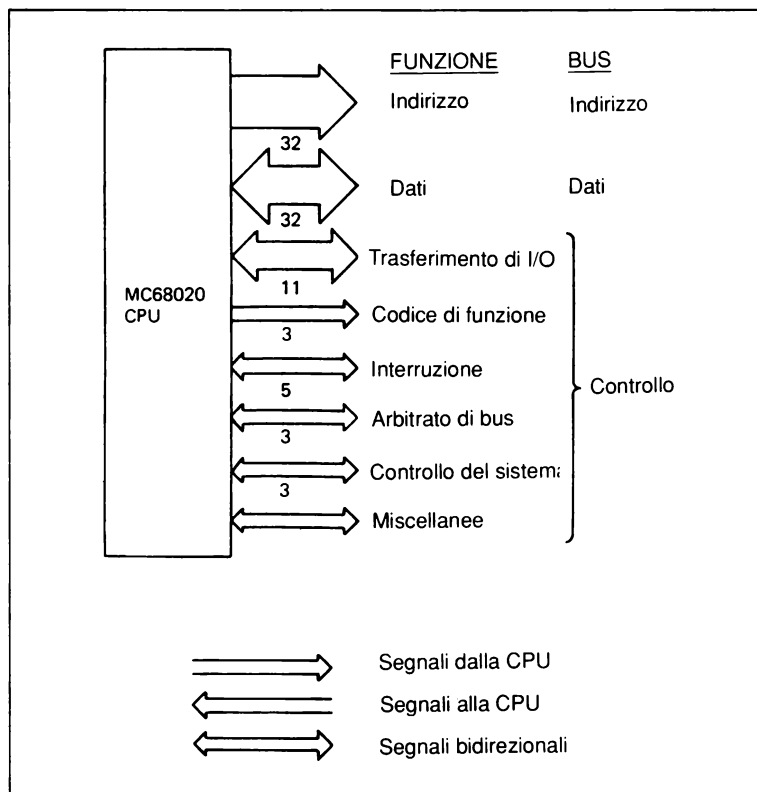



Fig. 2.12
Linee di segnale dell'MC68020.

trasferimento di I/O indicano se la locazione selezionata è di 8 bit (un byte), o di 16 bit (una word) o di 32 bit (una longword). Le 32 linee di dati trasferiscono i valori dei dati nell'una o nell'altra direzione, come indicato dalle doppie frecce (trasferimento bidirezionale). La maggior parte dei segnali di trasferimento di I/O avviano operazioni di lettura o scrittura del processore nella locazione indirizzata, mentre due linee consentono alla circuiteria esterna di riconoscere la richiesta del processore. Queste due linee indicano anche la dimensione in bit del bus di dati del dispositivo esterno. Ciò è noto come *dimensionamento dinamico del bus* e viene effettuato per ogni operazione di trasferimento. Un programmatore che intenda trasferire 8, 16 o 32 bit d'informazione da o verso un dispositivo periferico non ha bisogno di conoscere la dimensione del bus di dati per il dispositivo. Quindi un trasferimento di quattro byte (32 bit) ad un dispositivo di 8 byte avverrebbe automaticamente in quattro trasferimenti distinti.

Le linee del codice di funzione specificano la modalità (utente o supervisore) del processore durante il trasferimento dei dati. Esse servono anche a distinguere tra l'attività normale del programma e l'attività speciale del sistema. Per esempio, queste linee di segnale possono indicare il momento in cui la CPU sta riconoscendo un'interruzione o quando l'indirizzamento riguarda il coprocessore.

Le linee di controllo dell'interruzione consistono di tre segnali d'ingresso per determinare la priorità dell'interruzione (otto livelli) e di due altri segnali utilizzati per scopi speciali. Allorché un'interruzione viene riconosciuta, il controllo è trasferito ad una routine che gestisce l'interruzione. Quando più dispositivi possono richiedere un'interruzione al medesimo livello, la CPU o la circuiteria esterna deve determinare quale dispositivo dev'essere riconosciuto e servito per primo dalla routine d'interruzione pertinente al dispositivo.[11]

Le tre linee di controllo designate per l'arbitrato di bus consentono ad un circuito esterno di assumere il controllo del bus interno di sistema inviando una richiesta alla CPU. Allorché viene riconosciuto dall'MC68020, il processore viene isolato elettricamente dal bus. Tre altri segnali per il controllo del sistema sono usati per rivelare errori esterni o per consentire alla CPU di indicare il fallimento dell'operazione. Altri segnali miscellanei includono il segnale di clock e le connessioni per l'alimentazione elettrica.

Struttura dell'I/O e dell'interruzione. Poiché l'MC68020 e microprocessori simili sono tipicamente incorporati in sistemi complessi che richiedono la protezione della memoria ed un gran numero di unità periferiche, il processore è progettato per soddisfare tali requisiti. La Fig. 2.13 mostra un sistema di microcomputer in cui i segnali per la gestione della memoria, le richieste d'interruzione e le indicazioni di errore sono mostrati separatamente. Si può notare che il modo del processore (supervisore o utente) è impiegato dai circuiti di controllo della memoria per indirizzare l'area di memoria corretta. Se un programma nel modo di utente

[11] Come menzionato in un paragrafo precedente, fino a 192 dispositivi distinti possono condividere gli otto livelli di priorità, ma soltanto un dispositivo alla volta può essere riconosciuto ad un certo livello. Una volta che un dispositivo è stato riconosciuto, esso indica alla CPU quale delle 192 possibili routine di eccezione utilizzerà, come sarà spiegato ulteriormente nei capp. 11 e 13.

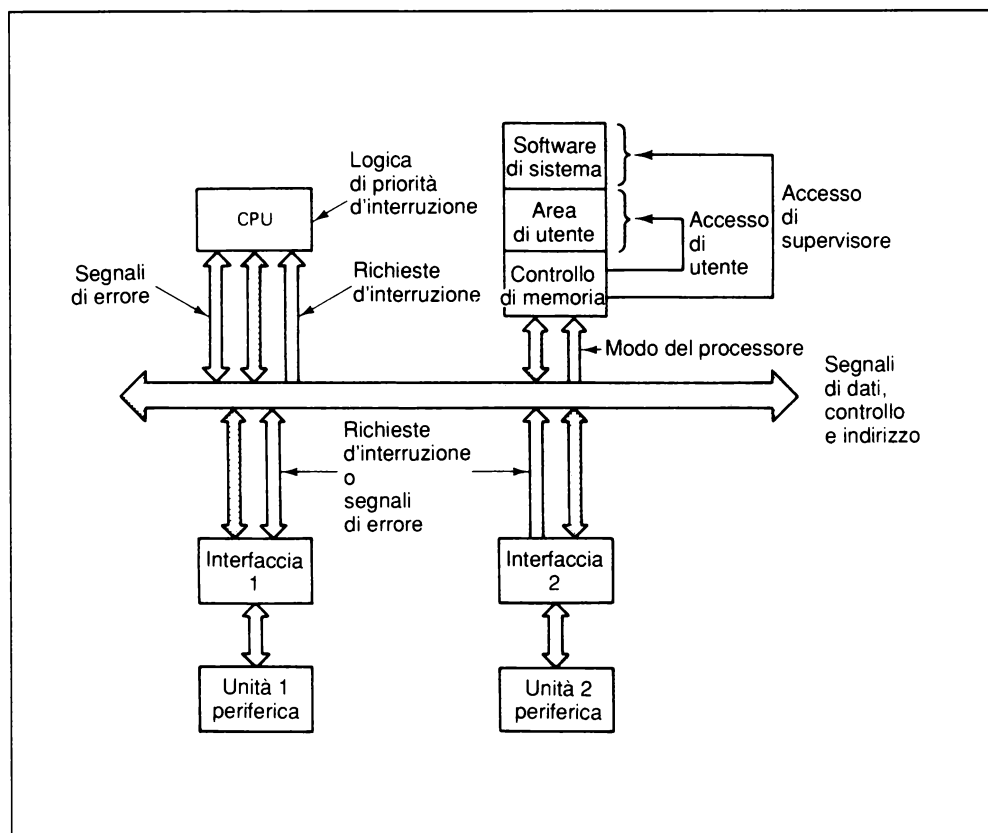



Fig. 2.13 *Struttura di bus di un sistema di microcomputer.*

tentasse di leggere o scrivere nell'area di supervisore della memoria, verrebbe generato un segnale di errore dalla circuiteria di controllo della memoria e sarebbe intrapresa un'azione appropriata da parte di una routine di errore eseguita dalla CPU.

Le linee di controllo di arbitrato del bus non sono mostrate esplicitamente nella Fig. 2.13. Queste tre linee determinano il dispositivo che dovrà assumere la funzione di master del bus di sistema, cioè quello che controllerà i trasferimenti di I/O ed operazioni simili. Tale arbitrato è richiesto quando più processori condividono il medesimo bus o quando un qualunque dispositivo diverso dalla CPU è in grado di avviare trasferimenti di I/O sul bus. I chip di accesso diretto alla memoria, discussi nel cap. 1, sono dispositivi di questo tipo.

I segnali di richiesta d'interruzione mostrati nella Fig. 2.13 sono elaborati dalla circuiteria d'interruzione della CPU. La CPU determina la priorità dell'interruzione e passa il controllo alla routine d'interruzione che corrisponde all'unità periferica di priorità più elevata che sta richiedendo il servizio. Al completamento di tale routine, il controllo passerà nuovamente al programma che era stato interrotto.

Controllo degli errori e debugging. Vari segnali di errore possono essere generati dalla circuiteria d'interfaccia mostrata in Fig. 2.13 o eventualmente dalla CPU stessa. L'MC68020 può inviare un segnale di errore allorché rivela che il sistema non può continuare ad operare correttamente a causa di un grave malfunzionamento esterno o di un guasto della CPU. Questa caratteristica può essere d'importanza capitale in un sistema multiprocessore in cui un processore guasto dev'essere isolato quando altri processori nel sistema rivelano un errore il cui effetto potrebbe ripercuotersi sull'intero sistema. Nel rivelare un malfunzionamento, la CPU segnala il problema su una delle tre linee di segnali di controllo del sistema; dopodiché, cessa di elaborare le istruzioni ed un altro dispositivo o CPU deve assumere il controllo del sistema.

Se viene indicato un errore di hardware, la CPU salva nella memoria le informazioni sulle condizioni del sistema relative al momento in cui l'errore è stato rivelato. Tali informazioni definiscono lo stato del processore, il tipo di operazione in corso, e simili dati riguardanti l'istante in cui si è verificato l'errore. Queste informazioni possono essere usate in certi casi per consentire al sistema di porre rimedio ad un errore dell'hardware durante la sua attività. Naturalmente, tali informazioni sono preziose anche per il progettista d'interfaccia per il debugging dell'hardware.

Esempio 2-3

Un impiego tipico della capacità di rivelazione degli errori dell'MC68020 è la determinazione dell'istante in cui un dispositivo esterno non risponde ad una richiesta di trasferimento di I/O. Una volta che la richiesta è stata effettuata, un circuito di temporizzazione nel sistema potrebbe indicare il tempo trascorso fino a quando il dispositivo non avrà riconosciuto la richiesta tramite le linee di controllo del trasferimento di I/O. Se nessun riconoscimento avviene entro un intervallo di tempo specificato (di solito, alcuni millisecondi), un segnale di errore, emesso da uno dei circuiti di temporizzazione "watchdog" (letteralmente: cane da guardia), sarebbe inviato al processore su una delle linee di segnali di controllo del sistema. A quel punto, l'azione successiva sarebbe decisa dall'apposita routine del processore per la gestione dell'errore. Se il malfunzionamento fosse grave, il processore potrebbe ritentare il trasferimento di I/O oppure segnalare un malfunzionamento del sistema su un'altra linea di controllo del sistema.

ESERCIZI

2.3.1 Si discutano le differenze tra una trappola ed un'interruzione. S'includano considerazioni sia sull'hardware che sul software.

2.3.2 Quali conseguenze potrebbe avere il fatto di permettere alla CPU di eseguire istruzioni non ammesse?

2.3.3 L'MC68020 richiede 26 cicli di clock per rispondere ad un'interruzione. A 20 MHz, ogni ciclo di clock è di 50 ns, per cui il tempo di risposta è di 1.3 microsecondi. Si supponga che tre routine d'interruzione richiedano il seguente tempo di esecuzione totale dopo il riconoscimento dell'interruzione:

R1 = 20 microsecondi
R2 = 30 microsecondi
R3 = 20 microsecondi

I livelli di priorità sono tali che R3 ha la massima priorità, mentre R1 ha quella minima. Qual è il possibile intervallo di tempo di esecuzione di ciascuna routine allorché si presenta l'interruzione corrispondente?

2.3.4 Si spieghi l'uso dei modi di supervisore e di utente. Si distinguano i tipi di programmi che vengono eseguiti nell'uno o nell'altro modo e se ne spieghino i motivi. Quali sono le istruzioni che potrebbero risultare limitate dai programmi eseguiti nel modo di utente?

2.3.5 Si discutano alcuni dei vantaggi e degli svantaggi della presenza di coprocessori rispetto alle medesime funzioni implementate direttamente sul chip della CPU.

2.3.6 Si discutano alcune applicazioni in cui sono richiesti sistemi multiutente con memoria virtuale. In quali applicazioni sono desiderabili sistemi multiprocessore?

2.3.7 Si discutano le modalità di traccia dell'MC68020. Qual è la finalità di questi ausilii al debugging?

CAPITOLO 3

RAPPRESENTAZIONI DI NUMERI E DI CARATTERI

Il computer digitale è in grado di memorizzare ed elaborare informazioni d'interesse per il programmatore. Le informazioni vengono registrate nella memoria come sequenze di cifre binarie che saranno elaborate dalla CPU. Per esempio, le istruzioni di linguaggio-macchina discusse nel cap. 2 rappresentano informazioni che controllano l'attività del sistema di computer. I *programmi*, che consistono di tali istruzioni, operano su altre sequenze binarie di *dati* che rappresentano le informazioni che sono state memorizzate per essere elaborate. Questo capitolo esplora i metodi di memorizzazione comunemente usati per rappresentare numeri e caratteri per sistemi di computer basati sull'MC68020.

I numeri che sono interpretati come interi positivi o negativi o come frazioni possono essere rappresentati nella memoria in molti modi. Il sistema numerico più comune adottato per rappresentare i numeri nei microcomputer è il sistema in *complemento a 2*, che rappresenta i numeri con segno come valori binari (cioè, in base 2). L'MC68020 dispone di istruzioni per l'addizione, la sottrazione, la moltiplicazione e la divisione di questi numeri binari. Questi numeri in complemento a 2 formano un *tipo di dati* fondamentale per l'MC68020.

Anche i numeri decimali possono essere sommati e sottratti con istruzioni dell'MC68020 se i valori decimali sono codificati in binario da un metodo noto come *decimale codificato in binario* (*Binary Coded Decimal:* BCD). Per comprendere sia numeri positivi che negativi, è impiegato il sistema in *complemento a dieci*.

Molte applicazioni scientifiche e tecniche richiedono un grande intervallo per i numeri che sono rappresentati in un formato di *virgola mobile:* si tratta di un equivalente binario della notazione scientifica, che utilizza una mantissa ed un esponente per rappresentare un numero, ed è impiegato — talvolta necessariamente — in sistemi basati sull'MC68020. Nessuna istruzione dell'MC68020 è disponibile per trattare direttamente con questi numeri. Tuttavia, la CPU MC68020 col suo coprocessore in virgola mobile dispone di istruzioni di questo tipo, per cui la programmazione risulterà notevolmente semplificata. In questo capitolo è discusso il formato di virgola mobile adottato nel sistema dell'MC68020, mentre una trattazione più esauriente sarà fornita nel cap. 12.

Un testo viene registrato nella memoria assegnando una specifica configurazione di bit a ciascun carattere dell'alfabeto. Il *codice ASCII* è il codice più noto per rappresentare i caratteri nei sistemi di microcomputer. Come avviene per i numeri in virgola mobile, qualsiasi elaborazione dei caratteri in codice ASCII viene effettuata tramite routine software, poiché l'MC68020 non ha istruzioni che operano direttamente sui caratteri.

In questo capitolo sono discusse le caratteristiche fondamentali dei tipi di dati comunemente impiegati nei sistemi basati sull'MC68020. Nei capitoli successivi saranno discusse le istruzioni di macchina per trattare i tipi di dati e saranno svolte varie altre considerazioni di programmazione. In particolare, le operazioni aritmetiche su interi saranno trattate in dettaglio nel cap. 7.

3.1 RAPPRESENTAZIONI DI NUMERI

In questo paragrafo si discute la rappresentazione di interi positivi e negativi e delle frazioni. Viene presentata una formulazione generale con la base o radice r per ogni rappresentazione di numeri; tale formulazione sarà quindi applicata alla discussione di valori binari con $r = 2$ e con altre basi a seconda dei casi. La presentazione generalizzata è utile per la conversione di numeri da una base all'altra e per le tecniche di analisi numerica. Sono presentate le rappresentazioni di numeri binari nei sistemi di segno e grandezza, in complemento a uno e in complemento a due. Inoltre sono presentate anche le rappresentazioni decimali nei sistemi in complemento a nove e in complemento a dieci.

3.1.1 Interi non negativi

Un intero non negativo in base r è scritto in notazione posizionale come segue:

$$N_r = (d_{m-1}\, d_{m-2} \cdots d_0)_r \qquad (3.1)$$

dove ogni cifra d_i può assumere uno dei valori distinti [0, 1, 2, ..., $r-1$], mentre m rappresenta la base 10 o il numero decimale di cifre nell'intero. Quindi il numero 324 avrebbe $d_0 = 4$, $d_1 = 2$ e $d_2 = 3$ nell'eq. (3.1) e potrebbe essere scritto come:

$$N_{10} = 324_{10}$$

Per i numeri in base 10, il pedice viene omesso se tale omissione non può dar luogo ad alcuna confusione. La forma specificata dall'eq. 3.1 è in genere denotata come notazione *posizionale*. La posizione della cifra, a partire dalla cifra più a destra, rappresenta una potenza della base r: cioè, 324 rappresenta 4 unità (4×10^0), 2 decine (2×10^1) e 3 centinaia (3×10^2). Matematicamente, il valore del numero è calcolato come segue:

$$N_r = d_{m-1}r^{m-1} + d_{m-2}r^{m-2} + \cdots + d_1 r + d_0 = \sum_{i=0}^{m-1} d_i r^i \tag{3.2}$$

in cui il valore delle cifre è limitato in modo che $0 \leq d_i \leq r-1$. Quindi il numero 324 può essere calcolato come:

$$324 = 3 \times 10^2 + 2 \times 10^1 + 4 \times 1$$

Le operazioni aritmetiche nell'eq. (3.2) potrebbero essere svolte in qualsiasi base numerica e questa equazione è usata spesso per determinare l'equivalente decimale di un numero in un'altra base.

La Tab. 3.1 elenca l'intervallo di possibili valori delle cifre nei sistemi numerici decimale, ottale e binario. Ovviamente, il sistema decimale è quello più usato dagli esseri umani per l'aritmetica ordinaria, mentre il sistema binario è quello più utilizzato per l'aritmetica nei computer. Le rappresentazioni ottale ed esadecimale sono comode per la scrittura di lunghi numeri binari. Per esempio:

$$01011010_2 = 5A_{16} = 132_8$$

Il valore decimale del numero rappresentato in queste basi è ottenibile dall'eq. (3.2) convertendo le cifre in base r negli equivalenti decimali, come segue:

$$N = 5 \times 16^1 + 10 \times 1 =$$

$$= 1 \times 8^2 + 3 \times 8^1 + 2 \times 1 =$$

$$= 90_{10}$$

Le cifre esadecimali [A, B, ..., F] rappresentano i numeri decimali [10, 11, ..., 15] nella conversione da esadecimale a decimale.

Tab. 3.1 *Cifre in vari sistemi di numerazione.*

Sistema numerico	**Base *r***	**Cifre**
Esadecimale	16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
Decimale	10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Ottale	8	0, 1, 2, 3, 4, 5, 6, 7
Binario	2	0, 1

Esempio 3-1

Si consideri il massimo intero positivo di m cifre in notazione posizionale:

$$N_r = ((r-1)(r-1)\cdots(r-1))$$

come in $(1111...1111)_2$ o in $(9999...9999)_{10}$, con m cifre ciascuno. La somma dell'eq. 3.2 indica che il valore decimale è:

$$N = (r-1)\sum_{i=0}^{m-1} r^i$$

che è una serie geometrica facile da sommare; il risultato è $r^m - 1$. Per esempio, un numero binario di 8 bit ha un valore massimo di $2^8 - 1$, cioè 255.

Valori frazionari positivi. La rappresentazione posizionale definita dall'eq. (3.1) è valida soltanto per gli interi. Se dev'essere rappresentata una frazione, s'impiega un punto di radice nella base r per separare la parte intera dalla parte frazionaria del numero.* Il punto di radice è noto come *punto binario* in base 2 e come *punto decimale* in base 10. Pertanto, 324.14 ha il valore:

$$3 \times 10^2 + 2 \times 10^1 + 4 \times 1 + 1 \times 10^{-1} + 4 \times 1^{-2}$$

In generale, una frazione positiva di k cifre viene scritta con un punto di radice significativo, come in:

$$.(d_{-1}\, d_{-2} \cdots d_{-k})_r \tag{3.3}$$

col valore:

$$.n_r = d_{-1}r^{-1} + d_{-2}r^{-2} + \cdots + d_{-k}r^{-k} \tag{3.4}$$

dove il pedice negativo delle cifre indica la potenza negativa appropriata di r.

Al suo interno, il processore esegue le operazioni aritmetiche sugli interi senza tener conto della posizione del punto di radice. È quindi possibile interpretare il valore interno di una frazione scrivendo $.n_r$ nella forma:

$$.n_r = r^{-k} \times (d_{-1}d_{-2} \cdots d_{-k}) \tag{3.5}$$

in cui l'espressione tra parentesi viene trattata come un valore intero. Per esempio, il numero $.1000_2$ ($= 0.5_{10}$) può essere scritto come:

$$2^{-4} \times (1000.)_2 = 2^{-4} \times 8$$

* *N.d.T.* Nella notazione italiana s'impiega la virgola al posto del punto.

che hanno entrambi il valore 0.5, come previsto. Il fattore di scala r^{-k} ha l'effetto di far scorrere il punti di radice di k posizioni a sinistra. Quindi $.n_r r^k$ può essere usato internamente come un operando intero e poi il risultato finale sarà scalato di r^{-k}. Per esempio, l'addizione dei valori binari .1000 (= 0.5_{10}) e .0100 (0.25_{10}) può essere eseguita come segue:

$$\begin{array}{rl} 1000. & \times 2^{-4} \\ +\underline{0100.} & \times 2^{-4} \\ 1100. & \times 2^{-4} \end{array}$$

Il risultato dell'addizione eseguita dalla macchina è 1100_2, cioè 12 in decimale, ed il programmatore deve applicare l'opportuno fattore di scala per ottenere il risultato aritmetico corretto:

$$12 \times 2^{-4} = 0.75$$

Oltre alla sottrazione, ogni valore scalato deve avere il medesimo fattore di scala. La scelta del fattore di scala può far sì che il punto di radice si trovi alla destra del numero (intero), a sinistra (frazione) o in un punto qualsiasi entro il numero. Quindi il valore 0.5 potrebbe essere espresso come una frazione in un codice binario di quattro cifre:

$$.1000_2$$

o come un intero, con un fattore di scala 2^{-4}, cioè:

$$(1000.)_2 \times 2^{-4}$$

o come una quantità mista scalata arbitrariamente; per esempio:

$$(10.00)_2 \times 2^{-2}$$

Quando il punto di radice è fissato per un particolare problema ed il programmatore deve tener conto dello scalamento, il sistema è noto come rappresentazione *a punto fisso*. Tutte le operazioni su interi con l'MC68020, come l'addizione e la sottrazione, presuppongono che il fattore di scala sia 2^0. Pertanto, il punto binario è a destra. L'importanza dell'eq. (3.5) sta nel fatto che, sia per l'analisi che per le operazioni di macchina, un valore frazionario dev'essere trattato come un intero durante tutti i passi intermedi di un calcolo. Il fattore di scala appropriato può essere applicato come ultimo passo, allorché si desidera ottenere l'effettivo valore numerico.

Esempio 3-2

Il primo esempio ha dimostrato che il massimo intero di m cifre per gli interi senza segno ha il valore $r^m - 1$. Quindi il massimo intero (binario) di 16 bit

$$1111\ 1111\ 1111\ 1111_2$$

ha il valore 2^{16}-1=65535 in rappresentazione decimale.
La più grande frazione a 16 bit

2^{-16} x 65535 = 0.99998474

Questo risultato è ottenibile scalando la frazione di 16 bit come segue:

$2^{-16} \times (2^{16} - 1) = 1 - 2^{-16}$

ed eseguendo l'aritmetica su una calcolatrice con un numero sufficiente di cifre decimali.

ESERCIZI

3.1.1.1 Si converta in decimale il seguente numero binario:
0100.0110_2

3.1.1.2 Qual è il valore decimale di
$1111\ 1111\ .\ 1111\ 1111\ 1111\ 1111_2$
con cinque cifre decimali nella frazione?

3.1.1.3 Si calcoli il valore decimale di ciascuno dei seguenti numeri:
(a) 130_9
(b) 120_5
(c) 0.7632_8
(d) $F00A_{16}$

3.1.1.4 Se
$111_x = 31_{10}$
qual è la base *x*?

3.1.1.5 Si determini il massimo intero rappresentabile in una word di computer di 32 bit. Si esprima il risultato come valore decimale.

3.1.2 Rappresentazioni di numeri con segno

Gli interi positivi, incluso lo zero, possono essere rappresentati comodamente come mostrato nel par. 3.1.1. Tuttavia, per rappresentare l'insieme completo dei numeri interi, che comprende gli interi positivi, lo zero e gli interi negativi, è necessaria una notazione per i valori negativi. Nell'aritmetica ordinaria, un numero negativo è rappresentato facendo precedere dal segno "meno" la grandezza (valore assoluto) del numero.Quindi –5 è un intero negativo di grandezza 5. Per i calcoli manuali, è comodo usare simboli distinti per denotare i numeri positivi (+) e negativi (–). Anche i circuiti aritmetici del computer che hanno il compito di trattare gli interi positivi e negativi risultano semplificati se s'impiega una delle cifre nella

notazione posizionale di un numero per indicare il segno dell'intero. Due di tali possibili rappresentazioni di interi con segno sono la notazione in *segno e grandezza* e la notazione in *complemento.* In entrambe le notazioni, il segno è indicato dalla cifra significativa più a sinistra nella forma posizionale del numero. Anche le frazioni negative possono essere rappresentate nell'uno e nell'altro sistema. Per le frazioni, la cifra del segno è scritta a sinistra del punto di radice.

Le istruzioni di aritmetica binaria dell'MC68020 operano direttamente solo su interi nella notazione in complemento a 2, se si considerano interi con segno. Gli interi in altre notazioni binarie o le frazioni devono essere trattati da programmi progettati a tal fine. Il trattamento di numeri decimali da parte della CPU è discusso nel par. 3.2, sebbene in questo paragrafo venga già introdotta la rappresentazione matematica di numeri decimali.

Rappresentazione in segno e grandezza. La rappresentazione in segno e grandezza di un numero nella notazione posizionale ha la forma seguente:

$$N_r = (d_{m-1}d_{m-2} \cdots d_1d_0)_r \tag{3.6}$$

in cui il segno del numero è indicato dalla cifra più significativa (più a sinistra):

$$d_{m-1} = \begin{cases} 0 & \text{se } N_r \geq 0 \\ r-1 & \text{se } N_r < 0 \end{cases} \tag{3.7}$$

Quindi, impiegando la rappresentazione in segno e grandezza, 1011_2 e 9003 sono numeri negativi di quattro cifre nei sistemi binario e decimale, rispettivamente. La grandezza del numero, scritta come $|N|$, è:

$$|N_r| = \sum_{i=0}^{m-2} d_i r^i \tag{3.8}$$

dove sono prese in considerazione soltanto le prime $m-1$ cifre a partire da destra. La versione positiva di un numero differisce da quella negativa soltanto per la cifra del segno, mentre le cifre $(d_{m-2}d_{m-3}\cdots d_1d_0)$ indicano la grandezza. Secondo le definizioni e in base all'eq. (3.8), i numeri di quattro cifre binarie 0011_2 e 1011_2 rappresentano i valori decimali 3 e –3, rispettivamente. Nella rappresentazione deve essere specificato il numero di cifre, inclusa la cifra del segno, altrimenti potrebbe esserci un'ambiguità d'interpretazione. Per esempio, si presume che 1011_2 in una rappresentazione di otto cifre diventi 00001011_2, che corrisponde al valore decimale 11. Per i valori binari, una frazione negativa nella notazione in grandezza e segno ha una cifra "di testa" uguale a 1, seguita dalla parte frazionaria. Quindi 1.100_2 rappresenta il numero decimale –0.5.

Esempio 3-3

Il numero 16 viene scritto in un sistema binario di 16 bit come:

$$0000\ 0000\ 0001\ 0000_2$$

Il numero −16 ha la seguente rappresentazione in segno e grandezza:

$$1000\ 0000\ 0001\ 0000_2$$

Rappresentazione in complemento. La maggior parte dei microprocessori, incluso l'MC68020, hanno istruzioni aritmetiche che operano su numeri negativi rappresentati in un sistema numerico di *complemento*.[1] In questi sistemi, i numeri positivi hanno la medesima rappresentazione della notazione in segno e grandezza, ma i numeri negativi sono formati calcolando il complemento del numero secondo le regole dello specifico sistema impiegato. I due più comuni sistemi in complemento utilizzati sono il *complemento alla radice* e il *complemento alla radice diminuito*. Dapprima sarà presentata la teoria generale di questi sistemi; dopodiché, saranno discussi i complementi a 2 e a 10 di numeri, come esempi di numeri in complemento alla radice. I complementi a uno e a nove sono esempi di sistemi in complemento alla radice diminuito per la base 2 e la base 10, rispettivamente.

In forma generale, il complemento alla radice di un numero di m cifre è calcolato matematicamente come:

$$N'_r = r^m - N_r \tag{3.9}$$

dove N'_r è il complemento alla radice del numero N_r in base r. Nei calcoli della macchina, si possono rappresentare soltanto valori di m cifre. Se un'operazione qualsiasi producesse un risultato che richiede più di m cifre, allora le cifre di ordine superiore sarebbero ignorate. Questa è una condizione di "fuori–intervallo". Un errore di macchina di questo tipo è noto come *overflow*.

I due sistemi in complemento alla radice usati con le istruzioni dell'MC68020 sono quelli in complemento a 2 e in complemento a 10. Il complemento a 2 di un numero N_2, ottenuto dall'eq. (3.9) usando 2 come base r, è dunque:

$$N'_2 = 2^m - N_2 \tag{3.10}$$

Quindi la forma di complemento a 2 con quattro cifre del numero −1 è:

[1] Le rappresentazioni in complemento hanno un vantaggio sulla notazione in segno e grandezza per il fatto che la cifra del segno non dev'essere trattata in modo speciale durante l'addizione e la sottrazione. Ciò semplifica alquanto i circuiti aritmetici della CPU, come discusso in vari riferimenti bibliografici relativi a questo capitolo, riportati nell'app. E.

$$N'_2 = 2^4 - 1 = 1\ 0000 - 0001 = 1111_2$$

Se il numero ed il suo complemento sono sommati:

$$\begin{array}{rrl} & 0001 & N \\ + & 1111 & N' \\ \hline 1 & 0000 & \end{array}$$

il risultato è 0 a quattro cifre, come dev'essere. In un sistema in complemento a 2, i valori negativi hanno sempre una cifra 1 di testa; invece, per i valori positivi, tale cifra è 0. Quindi + 4 = 0100_2, mentre –4 = 1100_2. Se viene eseguita un'addizione in una rappresentazione a quattro cifre su due numeri positivi, il risultato non deve essere maggiore di 7, altrimenti si verifica un overflow. Ciò limita l'intervallo dei numeri positivi di *m* bit al valore decimale $2^{m-1} - 1$. Nel caso di numeri di quattro bit, l'addizione di 4 + 5 in binario fornisce:

$$\begin{array}{rr} & 0100 \\ + & 0101 \\ \hline & 1001_2 \end{array}$$

che è un numero negativo nella notazione in complemento a 2. La sua grandezza, ottenibile dall'eq. 3.9, sarebbe:

$$N_2 = 2^4 - 1001 = 1\ 0000 - 1001 = 0111_2$$

cioè +7 in decimale, il che è chiaramente un errore. Nell'MC68020, viene fornita una segnalazione allorché si presenta tale condizione di overflow, per cui il programmatore deve prendere dei provvedimenti nel programma in vista di tali occorrenze.

Nel sistema in complemento a 10, in una rappresentazione con *L* cifre, il complemento a 10 di un numero *N* è formato come segue:

$$N' = 10^L - N \tag{3.11}$$

Nel caso di quattro cifre, –1 è rappresentato come:

$$N' = 10^4 - 1 = (10000 - 1) = 9999$$

L'MC68020 ha istruzioni aritmetiche per operare su numeri rappresentati nel sistema in complemento a 10.

Il complemento alla radice diminuito è calcolato come segue:

$$\overline{N}_r = r^m - N_r - 1 \tag{3.12}$$

che coincide col valore in complemento alla radice, ricavato dall'eq. (3.9), meno 1. Il complemento alla radice diminuito — o semplicemente "complemento", come viene chiamato di solito — è il complemento a 1 per valori binari e il complemento a 9 per numeri decimali. Il valore decimale di quattro cifre 0002 ha dunque il complemento:

$$\overline{N}_r = (10^4 - 0002) - 1 = 9999 - 0002 = 9997 \quad (3.13)$$

Aggiungendo 1 al complemento, si ottiene il complemento alla radice, come si può verificare confrontando le eqq. (3.12) e (3.9).

La Tab. 3.2 elenca il complemento alla radice per numeri binari di quattro cifre e decimali. Per un confronto sono presentati anche i valori in complemento a 1 e a 9. Si noti che nelle notazioni in complemento a 9 e a 10 i valori negativi hanno cifre di testa nell'intervallo da 5 a 9. La cifra del segno non è unica, come avviene nel caso dei valori negativi in complemento a 2.

Tab. 3.2 *Sistemi in complemento.*

Valore	Uno	Due	Valore	Nove	Dieci
7	0111	0111	4999	4999	4999
6	0110	0110	4998	4998	4998
5	0101	0101	.	.	.
4	0100	0100	.	.	.
3	0011	0011	.	.	.
2	0010	0010	0002	0002	0002
1	0001	0001	0001	0001	0001
0	0000	0000	0000	0000	0000
–0	1111	——	–0000	9999	——
–1	1110	1111	–0001	9998	9999
–2	1101	1110	–0002	9997	9998
–3	1100	1101	.	.	.
–4	1011	1100	.	.	.
–5	1010	1011	.	.	.
–6	1001	1010	.	.	.
–7	1000	1001	–4999	5000	5001
–8	——	1000	–5000	——	5000

Esempio 3-4

Il complemento alla radice di un numero si calcola facilmente complementando ciascuna cifra [sottraendola da $(r-1)$] e aggiungendo 1 al risultato formato dalle cifre complementate. Quindi il valore –2 è rappresentato come segue in vari sistemi a quattro cifre:

Complemento a 2: $-2 = (1111 - 0010) + 1 = 1110_2$
Complemento a 10: $-2 = (9999 - 0002) + 1 = 9998$
Complemento a 16: $-2 = (\text{FFFF} - 0002) + 1 = \text{FFFE}_{16}$

Esempio 3-5

Per una frazione la cui lunghezza è di k cifre, il complemento alla radice viene calcolato complementando ciascuna cifra e aggiungendo r^{-k} (non 1) al risultato. Pertanto, il numero

$$0101.01_2 = 5.25$$

ha come complemento il valore:

$$1010.10_2$$

La sua rappresentazione in complemento alla radice (cioè 2) sarebbe allora:

$$\begin{array}{r} 1010.10 \\ + \quad .01 \\ \hline 1010.11_2 \end{array}$$

dove è stato aggiunto il valore 2^{-2} al complemento a 1 del numero poiché $k = 2$. Similmente, la frazione 0.01_2 ha come complemento il valore 1.10_2 e come complemento a 2 il valore 1.11_2.

Intervallo numerico. L'intervallo di interi o frazioni per un certo sistema numerico è specificato mediante i valori minimo e massimo che tale sistema può rappresentare. Ad esempio, per interi positivi rappresentati con m cifre, ci sono r^m valori possibili in un intervallo numerico da 0 a $r^m - 1$. Per la rappresentazione con 8 bit di un intero binario positivo, l'intervallo è da 0 a $2^8 - 1$, o da 0 a 255. L'intervallo degli interi con segno in una rappresentazione con m cifre consente ancora r^m valori, ma metà di questi valori sono numeri negativi.

Il massimo intero positivo nella rappresentazione in segno e grandezza, o in complemento a 1 o in complemento a 2 è:

$$(0111...111)_2$$

dove compaiono ($m - 1$) cifre di valore 1. Il massimo valore decimale è pertanto $2^{m-1} - 1$, tenendo conto della discussione svolta nell'esempio 3.1. In una rappresentazione con 8 bit, il massimo numero positivo è $2^7 - 1$, cioè 127. Nella notazione in segno e grandezza, il valore più negativo è:

$$(1111...111)_2 = -(2^{m-1} - 1)$$

Il numero più negativo in complemento a 1 è $(100...00)_2$, che ha il medesimo valore decimale. Entrambi questi sistemi consentono sia un valore positivo che un valore negativo dello zero, poiché lo zero "positivo"

$$(000...000)_2$$

ha valori negativi $(1000...000)_2$ e $(1111...111)_2$ nelle notazioni in segno e grandezza e in complemento a 1, rispettivamente. Nella notazione in complemento a 2, tuttavia, è consentito un solo valore dello zero, poiché il complemento a 2 del numero 0 è il medesimo valore per gli m bit. Poiché ci sono in totale 2^m valori per ogni rappresentazione con m bit, la notazione in complemento a 2 consente un valore negativo in più rispetto alle altre.

Il complemento a 2 del valore positivo

$$(011...111)_2$$

è il numero negativo:

$$(100...001)_2 = -(2^{m-1} - 1)$$

per m bit. L'intero

$$(100...000)_2$$

deve rappresentare allora -2^{m-1} senza alcuna controparte positiva.

Il complemento a 10 di m cifre consente 10^m valori nell'intervallo

$$\text{da } -10^m/2 \text{ a } 10^m/2 - 1$$

come da −5000 a + 4999 nella rappresentazione con quattro cifre illustrata nella Tab. 3.2. I valori positivi sono:

$$0, 1, 2, ..., 499...99$$

per m cifre, mentre i valori negativi sono rappresentati come:

$$999...999, 999...998, ..., 500...001, 500...000$$

con valori −1, −2, ..., e così via. La rappresentazione in complemento a nove dei numeri negativi ha un numero 0 negativo e di conseguenza un valore negativo diverso da zero in meno rispetto a quelli consentiti nella notazione in complemento a dieci, cioè un intervallo:

$$\text{da } -10^m/2+1 \text{ a } 10^m/2-1$$

In questo caso, la grandezza del numero è ristretta dalla condizione:

$$|N| \leq 10^m/2-1$$

per cui le cifre più significative di 0, 1, 2, 3 o 4 indicano un numero positivo, mentre le cifre 5, 6, 7, 8 o 9 indicano un valore negativo.

Esempio 3-6

Il massimo numero positivo per un numero in complemento alla radice di m cifre in base r è:

$$(1/2) \times r^m - 1$$

poiché ci sono $(1/2) \times r^m$ interi positivi, incluso lo zero. Quindi il massimo valore in complemento a 2 è:

$$(1/2) \times 2^m - 1 = 2^{m-1} - 1$$

mentre il massimo valore del numero in complemento a 10 è:

$$(1/2) \times 10^m - 1$$

come indicato nella discussione precedente. Il numero di quattro cifre binarie consente valori fino a +7 nel sistema in complemento a 2, mentre il valore decimale di quattro cifre ha un massimo valore positivo di 4999.

Esempio 3-7

L'applicazione delle formule per i numeri più negativi e più positivi nella rappresentazione con m bit fornisce i risultati riportati nella tabella a pagina seguente.

RAPPRESENTAZIONE	PIÙ NEGATIVO	PIÙ POSITIVO
Segno e grandezza	$-2^{m-1}+1$	$2^{m-1}-1$
Complemento a 1	$-2^{m-1}+1$	$2^{m-1}-1$
Complemento a 2	-2^{m-1}	$2^{m-1}-1$

Per una rappresentazione con 16 bit, l'intervallo per le rappresentazioni in segno e grandezza e in complemento a 1 si estende da –32767 a +32767, mentre i numeri in complemento a 2 variano nell'intervallo –32768 a 32767.

Se è dotata di segno, una frazione binaria è rappresentata come:

$$(b_0.b_{-1}\cdots b_{-(k-1)})_2$$

mentre l'intervallo è determinato dal fattore di scala $2^{-(k-1)}$. Per esempio, la frazione di 8 bit nella rappresentazione in complemento a 2 ha l'intervallo da -1 a $1-2^{-7}$. I valori binari in questo intervallo sono:

```
1.000 0000        (-1)
1.000 0001
.
.
.
0.000 0000        (0)
.
.
.
0.111 1110
0.111 1111        (1 - 2^-7)
```

ESERCIZI

3.1.2.1 Si determini la rappresentazione in complemento a 2 dei seguenti numeri
(a) -0647_{16} con 16 bit
(b) -11_{10} con 16 bit
(c) -00101.110_2 con 8 bit

3.1.2.2 Il numero più negativo nel sistema in complemento a 2 è $100...0_2$ per *m* bit. Che valore si ottiene quando si effettua il complemento a 2 di tale numero?

3.1.2.3 Nella notazione in complemento a 2, il bit di segno ha il peso -2^{m-1} per un intero di m bit. Si determini il procedimento per estendere il numero di m bit a $2m$ bit per:
(a) un numero positivo;
(b) un numero negativo.
Tale procedimento è noto come *estensione di segno*.

3.1.2.4 Si rappresentino i numeri assegnati nella notazione specificata.
(a) Complemento a 9 di 653.72 con cinque cifre.
(b) -223_{16} nella forma in segno e grandezza, con quattro cifre esadecimali.
(c) $-3/8$ nella forma in complemento a 1 con 8 bit, incluso il bit di segno.

3.1.2.5 Si determini l'intervallo di numeri per le forme in segno e grandezza, complemento a 1 e complemento a 2 per una rappresentazione con m bit se:
(a) $m = 8$
(b) $m = 16$
(c) $m = 32$

3.1.2.6 Se il massimo numero positivo in una rappresentazione in complemento a 10 con quattro cifre è limitato a 999 (cioè, a tre cifre), si determinino l'intervallo e la rappresentazione corrispondenti dei numeri negativi. Si noti che i numeri negativi iniziano sempre con 9 come cifra del segno, quando la grandezza di tali numeri è limitata in questo modo.

3.1.3 Conversioni tra le rappresentazioni

I sistemi numerici di maggior interesse per gli utenti di computer sono quello binario, l'ottale, il decimale e l'esadecimale. Sebbene la rappresentazione interna alla macchina nei microcomputer sia quella binaria, le altre rappresentazioni sono importanti per la comodità dell'utente. Questo è il motivo per cui sono frequentemente richieste le conversioni di numeri dal sistema decimale ad altre basi e viceversa.

Talvolta sono necessarie conversioni tra basi numeriche arbitrarie, anche se quelle di maggior importanza nel lavoro col computer sono le conversioni tra i sistemi binario, ottale ed esadecimale. Per fortuna, le conversioni tra queste basi sono immediate.

Conversione in decimale di numeri positivi. Il numero N_r in base r può essere rappresentato in decimale come:

$$N.n = D_{m-1}r^{m-1} + \cdots + D_0 + D_{-1}r^{-1} + \cdots + D_{-k}r^{-k} \qquad (3.14)$$

dove i D_i sono i valori equivalenti in base 10 delle cifre in base r. Il numero viene convertito moltiplicando ciascuna cifra per l'appropriata potenza di r e aggiungendo ciascun risultato alla somma. Il numero è stato designato qui come $N.n$ per evidenziare il fatto che esso ha sia una parte intera che una parte frazionaria.

Esempio 3-8

Per convertire 11 1110_2 in decimale, il valore viene calcolato come somma di una serie dall'eq. (3.2) o (3.14), ottenendo il risultato seguente:

$$\begin{aligned} N &= 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 = \\ &= 32 + 16 + 8 + 4 + 2 = \\ &= 62 \end{aligned}$$

Esempio 3-9

Il valore di 0.502_8 in decimale è:

$$\begin{aligned} 0.n &= 5 \times 8^{-1} + 0 + 2 \times 8^{-3} = \\ &= 0.6250 + 0.003906250 = \\ &= 0.62890625_{10} \end{aligned}$$

come si può determinare dall'eq. (3.4) o (3.14).

Conversione da decimale in una base numerica arbitraria. Per convertire manualmente un numero decimale ad un numero in una base diversa, è comodo operare nel sistema decimale con la rappresentazione in serie del numero. La conversione di un intero positivo viene effettuata tramite divisione ripetuta per la nuova radice usando i resti in successione come cifre nel nuovo sistema. Una frazione viene convertita mediante moltiplicazione ripetuta per la radice; in questo caso, le cifre del risultato sono costituite dalle parti intere dei prodotti.

Esempio 3-10

Per convertire 3864_{10} in ottale, il numero viene diviso ripetutamente per 8, come segue:

$$\begin{aligned} 3964/8 &= 495 + (4/8) \\ 495/8 &= 61 + (7/8) \\ 61/8 &= 7 + (5/8) \\ 7/8 &= 0 + (7/8) \end{aligned}$$

I resti di ciascuna divisione, rappresentati dai numeratori di ciascuna frazione, sono le cifre del risultato. L'ordine di tali cifre è opposto quello in cui sono state ottenute. Quindi, nell'esempio precedente, si ha:

$$3964_{10} = 7574_8$$

Esempio 3-11

Il numero 0.78125_{10} viene convertito in una frazione esadecimale come segue:

$$0.78125 \times 16 = 12.0 + 0.5$$
$$0.5 \times 16 = 8.0 + 0$$

Il risultato è pertanto $0.78125_{10} = .C8_{16}$.

Per comprendere la teoria di queste conversioni, si scriva l'equazione 3.14 nella forma:

$$N = ((\cdots((d_{m-1} r + d_{m-2})r + d_{m-3})r + \cdots + d_1)r + d_0)$$

per la parte intera e la si eguagli al valore decimale che rappresenta. Dopodiché si divida N per la base desiderata. Usando 8 come base nell'esempio 3.10, il primo resto è il valore ottale d_0. Continuando a dividere i numeri globali (quozienti) per la base, si ottiene in successione d_0, d_1, d_2, ..., d_{m-1}, in quest'ordine. Trattando in questo modo la parte frazionaria dell'eq. (3.14) con potenze negative di r, si può comprendere come è stato calcolato il valore nell'esempio 3.11.

Conversione di un numero da una base arbitraria ad un'altra. Un numero scritto in una base r_1 può essere convertito ad un numero in una base r_2 eseguendo operazioni aritmetiche in una base diversa da quella decimale. Per i calcoli a mano, un metodo più comodo consiste nel convertire il numero selezionato in forma decimale dalla base r_1 e poi convertire il risultato dal sistema decimale alla base r_2.

Esempio 3-12

La conversione di 112_3 alla base 5 viene effettuata convertendo dapprima 112_3 in decimale:

$$1 \times 3^2 + 1 \times 3^1 + 2 = 9 + 3 + 2 = 14_{10}$$

Dopodiché il numero decimale 14 viene convertito alla base 5 nella forma:

$$\begin{aligned} 14/5 &= 2 + (4/5) \\ 2/5 &= 0 + (2/5) \end{aligned}$$

cioè $14_{10} = 24_5$. Quindi $112_3 = 24_5$.

Conversioni di numeri positivi con basi che sono potenze di 2. Se la relazione tra una base di numerazione r_1 ed una base di numerazione r_2 è della forma:

$$r_2 = r_1^L$$

dove L è un intero positivo o negativo, allora la conversione tra le basi risulta particolarmente semplice se s'impiega la notazione posizionale. In particolare, poiché le basi binaria, ottale ed esadecimale sono legate dalle relazioni:

$$\begin{aligned} 16 &= 2^4 \\ 8 &= 2^3 \end{aligned}$$

la conversione da binario a ottale o da binario a esadecimale richiede soltanto il raggruppamento delle cifre binarie in gruppi di tre o di quattro, rispettivamente. La conversione da ottale a binario o da esadecimale a binario richiede che ciascuna cifra ottale o esadecimale sia sostituita dal suo equivalente binario.

La conversione tra numeri ottali ed esadecimali può essere effettuata semplicemente usando la rappresentazione binaria come passo intermedio, poiché le basi 8 e 16 non sono in relazione diretta.

Esempio 3-13

La conversione in ottale del numero binario 1011 0111.0010 1_2 richiede il raggruppamento delle cifre a tre a tre, a partire dalla cifra binaria meno significativa (più a destra) per la porzione intera, mentre per la parte frazionaria si deve partire dalla cifra più significativa (più a sinistra). Quindi la conversione viene effettuata come segue:

$$(010)\ (110)\ (111)\ .\ (001)\ (010) = 267.12_8$$

in cui sono state aggiunte cifre binarie zero a ciascuna estremità per formare le cifre ottali prima della conversione.

Conversione di numeri negativi da binario a decimale. La conversione in decimale di un numero binario positivo si effettua facilmente col metodo delle serie di potenze illustrato precedentemente. Si può ottenere in questa maniera anche la conversione di numeri negativi nella notazione in complemento a 1 o in complemento a 2, purché al bit di segno sia assegnato l'appropriato valore o peso decimale. Come esempio, si considerino i seguenti numeri negativi nella rappresentazione in complemento a 2 con 8 bit ed i rispettivi equivalenti decimali:

$$1111\ 1111_2 = -1$$
$$1111\ 1110_2 = -2$$

.
.
.

$$1000\ 0001_2 = -127$$
$$1000\ 0000_2 = -128$$

Associando la cifra di testa con -2^7 (= −128) e sommando i valori posizionali positivi delle restanti cifre, si ottiene il valore decimale appropriato. Quindi il valore decimale di un numero negativo di 8 bit nella notazione in complemento a 2 è:

$$-2^7 + d_6 \times 2^6 + d_5 \times 2^5 + \cdots + d_0$$

quando il numero negativo nella sua forma posizionale è:

$$(1 d_6 d_5 d_4 \cdots d_0)_2$$

Esaminando i valori positivi, il caso generale per numeri positivi e negativi in complemento a 2 può essere ricavato per calcolare l'equivalente decimale come:

$$N = -d_{m-1} \times 2^{m-1} + \sum_{i=0}^{m-2} d_i \times 2^i \qquad (3.15)$$

con d_{m-1} = 0 per un valore positivo o d_{m-1} = 1 per un valore negativo. Da questa equazione, il numero di 8 bit $1000\ 0010_2$ ha il valore decimale:

$$N = -2^7 + 0 + \cdots + 1 \times 2^1 + 0 = -126$$

con bit di segno $d_7 = 1$. Il numero $0000\ 0010_2$, con $d_7 = 0$, ha il valore:

$$-0 \times 2^7 + 0 + \cdots + 1 \times 2^1 + 0 = +2$$

In sostanza, l'eq. 3.15 rappresenta una notazione compatta per il calcolo del valore decimale di un numero di m bit nel sistema in complemento a 2. I pesi decimali della cifra di testa per gli interi in complemento a 1 e in complemento a 2 e le frazioni sono elencati nella Tab. 3.3. La grandezza di un numero in segno e grandezza si ottiene semplicemente moltiplicando per +1 o per −1 a seconda del segno.

Tab. 3.3 *Valori del bit di segno.*

Rappresentazione	**Peso in decimale**	
	Intero (m bit)	*Frazione* (k bit con segno)
Complemento a 1	$1 - 2^{m-1}$	$2^{-(k-1)} - 1$
Complemento a 2	-2^{m-1}	-1

Esempio 3-14

(a) L'intero $1000\ 0011_2$ nella notazione in segno e grandezza ha il valore:

$$(-1)(0 \times 2^6 + \cdots + 1 \times 2^1 + 1) = -3_{10}$$

poiché il modulo è moltiplicato per −1.

(b) Il numero $1111\ 1001_2$ nella notazione in complemento a 1 ha il valore:

$$(1 - 2^7) + (1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1) =$$
$$= -127 + 121 = -6_{10}$$

usando per il bit di testa il peso mostrato nella Tab. 3.3.

(c) Il numero in complemento a due $111\ 1001_2$ ha il valore:

$$-2^7 + (1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1) =$$
$$= -128 + 121 = -7_{10}$$

Esempio 3-15

(a) La frazione $1.001\ 0000_2$ nella notazione in segno e grandezza ha il seguente valore:

$$(-1)(0 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}) = -0.125_{10}$$

(b) La frazione $1.100\ 1111_2$ nella notazione in complemento a 1 ha il valore:

$$(2^{-7} - 1) + 1 \times 2^{-1} + 1 \times 2^{-4} + 1 \times 2^{-5} + 1 \times 2^{-6} + 1 \times 2^{-7}) =$$
$$= -0.375_{10}$$

ESERCIZI

3.1.3.1 Si convertano i seguenti numeri come indicato:
(a) 1024_{10} in binario
(b) 53000_{10} in esadecimale
(c) FFFF FFFF_{16} in decimale
(d) 35_{10} alla base 5.

3.1.3.2 Si converta in decimale la frazione ottale periodica $(0.333...)_8$.

3.1.3.3 Si dimostri che aggiungendo 1 alla forma in complemento del numero positivo N si ottiene la rappresentazione in complemento alla radice $r^m - |N|$ per un numero di m cifre.

3.1.3.4 Si converta in decimale la frazione $1.111\ 1111_2$ espressa nella forma in complemento a 2.

3.1.3.5 Usando la rappresentazione in complemento a 2, si rappresentino i numeri nell'intervallo $-2, -1^3/_4, ..., 1^1/_2, 1^3/_4$.

3.2 DECIMALE CODIFICATO IN BINARIO

Molti microcomputer dispongono di istruzioni per eseguire operazioni aritmetiche su dati considerati come numeri decimali. Ciò è comodo per l'elaborazione aziendale e nella rappresentazione di dati che sono intrinsecamente decimali. Ad esempio, i selettori rotanti possono presentare il dato di uscita come una cifra decimale codificata in binario, per rappresentare la cifra selezionata sulla manopola. Molti visualizzatori sono progettati per ricevere cifre decimali in codice e visualizzare il risultato in decimale.

Un sistema di codifica decimale è il decimale codificato in binario (*Binary Coded Decimal:* BCD). Nel sistema BCD, i primi 10 numeri binari corrispondono a cifre decimali. Esso è talvolta denotato come il sistema "naturale" di decimale codificato in binario.

In questo paragrafo sono discussi il sistema decimale codificato in binario e le varie operazioni di conversione con numeri BCD. L'impiego della notazione in complemento a 10 è comoda per rappresentare valori BCD negativi, sebbene siano possibili altre rappresentazioni. Qui viene discusso soltanto il complemento a 10, poiché questo è il metodo adottato per l'addizione e la sottrazione di valori decimali con segno eseguiti dal processore MC68020.

3.2.1 Rappresentazione in BCD di interi positivi

In molte applicazioni, particolarmente quelle che riguardano transazioni finanziarie, è desiderabile una rappresentazione effettiva di numeri decimali in una macchina che opera su cifre binarie. Poiché una qualsiasi cifra decimale può essere rappresentata da quattro cifre binarie, è naturale scegliere un codice binario in cui sono impiegate quattro cifre binarie per ciascuna cifra decimale. Un codice siffatto è mostrato nella Tab. 3.4, che elenca i valori della rappresentazione decimale codificata in binario (BCD). I valori binari possibili da 1010_2 a 1111_2 non sono usati, poiché i valori decimali da 10 a 15 richiedono due cifre BCD per poter essere rappresentati.

Tab. 3.4 *Valori decimali codificati in binario.*

BCD	BINARIO
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Ogni cifra decimale ha il valore:

$$D_i = b_{i3} \times 2^3 + b_{i2} \times 2^2 + b_{i1} \times 2^1 + b_{i0} \quad (3.16)$$

dove b_{ij} è la j-esima cifra binaria nella rappresentazione della i-esima cifra decimale. Il valore del numero BCD di L cifre è calcolato in decimale come:

$$N = D_{L-1} \times 10^{L-1} + D_{L-2} \times 10^{L-2} + \cdots + D_0 \quad (3.17)$$

dove ogni D_i è formato come indicato dall'eq. 3.16. Per esempio, il valore decimale 95 sarebbe codificato in binario come:

$$1001\ 0101_2$$

ed è memorizzato in questa forma. Usando l'eq. 3.16, si ha:

$$D_0 = 1 \times 2^2 + 1 = 5$$

e

$$D_1 = 1 \times 2^3 + 1 = 9$$

I numeri in BCD possono essere sommati o sottratti dalle istruzioni dell'MC68020 che eseguono aritmetica decimale. Pertanto, il programmatore non deve preoccuparsi della rappresentazione interna. Il formato interno dev'essere preso in considerazione soltanto nel caso in cui siano richieste delle conversioni tra i numeri BCD ed altre rappresentazioni.

Esempio 3-16

Il numero binario

0001 0111 0011 1001

ha il valore BCD

$$N = 1 \times 10^3 + 7 \times 10^2 + 3 \times 10^1 + 9 = 1739$$

Le 16 cifre binarie codificate come numeri positivi BCD hanno un intervallo limitato a $0 \leq N \leq 9999$.

Esempio 3-17

I microprocessori che eseguono operazioni aritmetiche su operandi di 8 bit (byte) e di lunghezze maggiori possono consentire tali operazioni su interi BCD "impaccati", come avviene nell'MC68020. In questa rappresentazione, due cifre BCD sono contenute in ciascun valore di 8 bit, anziché memorizzare ogni cifra BCD in un byte distinto (la cosiddetta *notazione BCD non impaccata*).

Il numero BCD 3475 può essere trattato per gli scopi dei calcoli di macchina sia come BCD impaccato che come BCD non impaccato, nella maniera seguente:

VALORE DECIMALE		MEMORIA (BINARIO)
Impaccato	34	0011 0100
	75	0111 0101
Non impaccato	03	0000 0011
	04	0000 0100
	07	0000 0111
	05	0000 0101

Nella rappresentazione non impaccata, la singola cifra è mostrata in una locazione di byte con una cifra 0 di testa, poiché l'MC68020 e la maggior parte dei processori indirizzano locazioni di memoria contenenti 8 bit. Il formato non impaccato è tipicamente usato quando sono impiegati algoritmi per eseguire la moltiplicazione o la divisione di numeri BCD. Le istruzioni dell'MC68020 per impaccare e disimpaccare numeri BCD saranno presentate nel cap. 7.

ESERCIZI

3.2.1.1 Si mostri la rappresentazione interna di macchina (binaria) dei seguenti numeri positivi nel formato BCD impaccato:
(a) 07
(b) 13
(c) 99

3.2.1.2 Si determinino i valori decimali dei seguenti numeri positivi codificati nel formato BCD:
(a) $0001\ 1001\ 0111\ 0000_2$
(b) $0001\ 1111_2$

3.2.1.3 Assumendo che la rappresentazione sia in BCD impaccato, si calcoli l'intervallo decimale per la rappresentazione di BCD positivi, usando:
(a) 8 bit
(b) 16 bit
(c) 32 bit

3.2.1.4 Si descriva il test richiesto per garantire che una condizione di fuori–intervallo venga rivelata quando sono sommati (sottratti) due interi positivi BCD. Si assuma che la massima lunghezza sia di L cifre decimali per ogni intero BCD. L'MC68020 è dotato di hardware incorporato (codici di condizione) per rivelare tali condizioni.

3.2.2 Conversione tra BCD e binario

Quando sono richieste rappresentazioni di macchina come sequenze binarie, è comodo l'impiego dell'aritmetica in base 2 anziché in base 10 con l'MC68020. Ciò consente ad un programma di trarre vantaggio delle sue istruzioni di moltiplicazione e divisione per eseguire l'aritmetica binaria. Quindi la conversione del numero positivo BCD

$$(D_{L-1} D_{L-2} \cdots D_0)$$

può essere effettuata scrivendo dapprima l'eq. 3.17 nella forma:

$$N = (\cdots((D_{L-1}) \times 10 + D_{L-2}) \times 10 + \cdots + D_1) \times 10 + D_0$$

Convertendo tutte le cifre in binario, si ottiene un'equazione utile per l'implementazione di macchina. Il valore binario del numero BCD è allora:

$$N_2 = (\cdots((D_{L-1} \times 1010_2 + D_{L-2}) \times 1010_2 + \cdots + D_1) \times 1010_2 + D_0$$

dove 1010_2 è 10 in decimale e le cifre D_i sono espresse nella loro forma binaria di 4 bit. Dapprima la cifra più significativa viene moltiplicata per 1010_2, poi viene aggiunta la cifra successiva in ordine di significatività e la somma viene moltiplicata per 1010_2, e così via, finché viene aggiunta l'ultima cifra D_0. Il risultato è una rappresentazione binaria con m cifre del numero BCD. La programmazione dell'equazione di conversione è semplice utilizzando l'insieme di istruzioni dell'MC68020. I valori BCD vengono talvolta convertiti in binario per l'elaborazione di macchina, poiché la CPU ha un insieme esteso di istruzioni che operano su numeri binari, ma relativamente poche istruzioni per trattare interi BCD.

Quando è necessario convertire numeri binari nella rappresentazione BCD, la conversione viene effettuata tramite divisione ripetuta per 10 (1010 in binario). Ogni resto è una cifra BCD a partire dalla cifra meno significativa. Ciò viene effettuato di solito prima che i numeri binari interni siano presentati in uscita per essere visualizzati come un valore decimale.

Esempio 3-18

Impiegando l'aritmetica in base 2, il numero 99 in BCD viene convertito in binario come segue:

$$N_2 = 1001 \times 1010 + 1001 = 0110\ 0011$$

Esempio 3-19

Il numero binario 0010 0100_2 viene convertito nella rappresentazione di macchina dell'equivalente BCD impiegando l'aritmetica binaria per la rappresentazione, come segue:

$$\frac{0010\ 0100}{1010} = 0011 + (0110)$$

$$\frac{0011}{1010} = 0 + (0011)$$

Qui i resti tra parentesi rappresentano la sequenza binaria:

0011 0110

che è intepretata come l'equivalente decimale 36.

ESERCIZI

3.2.2.1 Si converta 1000 0000_2 in BCD tramite divisione ripetuta per 1010_2 in binario.

3.2.2.2 Si converta il numero BCD 509 in binario usando sia l'aritmetica in base 10 che l'aritmetica in base 2.

3.2.2.3 Si consideri la moltiplicazione di numeri nella rappresentazione BCD. Si definisca un algoritmo per moltiplicare un valore BCD a più cifre per un moltiplicatore di una singola cifra, assumendo che sia possibile soltanto la moltiplicazione binaria e che le cifre siano in forma non impaccata. Si supponga altresì che sia disponibile un'istruzione di addizione BCD per sommare i risultati parziali.

3.2.3 Interi negativi BCD

L'MC68020 ha istruzioni per eseguire l'aritmetica su numeri BCD rappresentati nella notazione in complemento a 10. Come descritto nel par. 3.1, il complemento a 10 di un numero decimale *N è formato da:*

$$N' = 10^L - N \tag{3.18}$$

quando sono impiegate *L* cifre per rappresentare il numero. Per i calcoli con carta e penna, il complemento a 10 si ottiene facilmente complementando il numero cifra per cifra (complemento a nove) e aggiungendo 1 al risultato.

Esempio 3-20

Il complemento a dieci di 1319 in una rappresentazione con cinque cifre è:

$$N' = 100000 - 1319 = (99999 - 1319) + 1 = 98681$$

La rappresentazione BCD nella memoria sarebbe:

$$1001\ 1000\ 0110\ 1000\ 0001_2$$

Esempio 3-21

Il complemento a 10 di un numero può essere formato anche sottraendolo da 0 ed ignorando il riporto nella cifra più significativa. Quindi il complemento a dieci di 98681 è:

$$0 - 98681 = 01319$$

come mostrato. L'istruzione NBCD (negazione di decimale) dell'MC68020 svolge questa operazione per formare il complemento a 10 di un numero.

ESERCIZI

3.2.3.1 Si determini la rappresentazione (binaria) di macchina dei seguenti numeri BCD con segno, con una lunghezza di word di 16 bit:
(a) 124
(b) –1
(c) –1000
(d) 5024

3.2.3.2 Qual è l'intervallo di un numero BCD con segno che può essere rappresentato da m cifre binarie per:
(a) $m = 8$
(b) $m = 16$
(c) $m = 32$
quando il valore positivo è ristretto al valore massimo di $10^L - 1$ per L cifre decimali?

3.2.3.3 Si dimostri che il complemento a 9 di una cifra BCD può essere formato aggiungendo 6 e poi formando il complemento a 1 del risultato in notazione binaria.

3.3 RAPPRESENTAZIONE IN VIRGOLA MOBILE

Nelle rappresentazioni di numeri considerate in precedenza si presupponeva che il punto di radice fosse situato in una posizione fissa, fornendo un intero o una frazione come interpretazione della rappresentazione interna di macchina. Sarebbe dunque il programmatore ad avere la responsabilità di variare la scala degli operandi numerici in modo da adattarli alla lunghezza di word selezionata e infine di variare la scala in senso inverso per ottenere i risultati corretti. Naturalmente il punto di radice non viene effettivamente memorizzato col numero, ma la sua posizione dev'essere ricordata dal programmatore. Questo è il cosiddetto metodo di rappresentazione in *virgola fissa.*

In pratica, il valore di macchina è limitato ad un intervallo finito, determinato dal numero di cifre binarie impiegate nella rappresentazione. Per una word di 32 bit, l'intervallo degli interi in virgola fissa con segno è di $+2^{31}$, cioè circa $+10^9$. Quindi l'intervallo limitato della notazione in virgola fissa è un inconveniente per certe applicazioni. Inoltre, le unità arimetiche che operano su numeri in virgola fissa non hanno in genere la capacità di arrotondare i risultati. Come discusso in vari riferimenti bibliografici relativi a questo capitolo nell'app. E del libro, ciò limita l'utilità della notazione in virgola fissa nel calcolo scientifico.

Per superare molte limitazioni della notazione in virgola fissa, nei sistemi digitali si adotta per i numeri una notazione che è una controparte di quella scientifica. La notazione in *virgola mobile* rappresenta un numero come una parte frazionaria per una base selezionata elevata ad una potenza. Nella rappresentazione di macchina, sono memorizzati soltanto la parte frazionaria ed il valore dell'esponente. L'equivalente decimale è scritto come:

$$N.n = f \times r^e \qquad (3.19)$$

dove f è la frazione o *mantissa,* mentre e è un intero positivo o negativo denominato *esponente.* Di solito viene scelta la base 2, sebbene sia talvolta impiegata la base 16.

Un certo numero di opzioni sono offerte al progettista di un formato in virgola mobile. Ciò vale sia nel caso in cui le operazioni aritmetiche sono eseguite direttamente dalla CPU o dal suo coprocessore, sia nel caso in cui tali operazioni vengano eseguite da un "pacchetto" di software contenente routine per l'aritmetica in virgola mobile. Il numero di formati distinti è esorbitante e solo di recente è stato fatto un tentativo di normalizzare l'aritmetica in virgola mobile per i computer.

Lo standard proposto dall'IEEE è stato adottato dalla Motorola per un certo numero dei suoi prodotti. Questo standard IEEE descrive con precisione i formati ed altri aspetti dell'artimetica in virgola mobile necessari affinché il computer operi coerentemente anche quando le operazioni vengono eseguite su sistemi diversi.

3.3.1 Formati in virgola mobile

Il formato tipico in virgola mobile memorizza insieme la frazione e l'esponente in una rappresentazione con *m* bit. La scelta per un formato in virgola mobile di lunghezza fissa è comunemente di 32 o 64 bit, denotati rispettivamente come "singola precisione" e "doppia precisione". Talvolta sono impiegati formati estesi con $m > 64$, nei casi in cui sono richiesti un intervallo più ampio o una precisione maggiore.

Una volta che è stata scelta la lunghezza della rappresentazione in virgola mobile, è possibile un certo numero di scelte sia per la lunghezza che per il formato della frazione e dell'esponente. Poiché sia la frazione che l'esponente o entrambi potrebbero essere positivi come pure negativi, è necessario che entrambi siano dotati di segno. Infine, l'intepretazione dei bit all'interno della rappresentazione in virgola mobile dipende dalle posizioni della frazione e dell'esponente.

Molti formati in virgola mobile impiegano una rappresentazione di segno e grandezza per la frazione. Il bit più significativo della word è riservato al segno, il che rende più facile determinare se il numero in questione è positivo o negativo. La frazione è generalmente normalizzata, per contenere il massimo numero possibile di cifre significative. Pertanto, in un sistema in base *r*, la cifra più significativa si trova nella posizione più a sinistra nella frazione. Per numeri non nulli nel sistema binario, la cifra più a sinistra sarà un 1. Quando l'unità aritmetica o il programma fa scorrere le cifre nella frazione durante le operazioni aritmetiche, l'esponente viene "aggiustato" di conseguenza. Quando è normalizzata come un valore in base 2, la grandezza della frazione è:

$$0.5 \leq |f| < 1 \tag{3.20}$$

a meno che il numero sia zero. Il numero di cifre riservate per la frazione rappresenta un compromesso tra la precisione della frazione e l'intervallo dell'esponente. Un tipico formato in singola precisione (32 bit) potrebbe contenere un esponente di 8 bit ed una frazione di 23 bit, escludendo il segno.

Un esponente potrebbe essere rappresentato nel complemento a 2 o in qualsiasi altra notazione che ammetta valori con segno. Un'alternativa diversa, che consente una rappresentazione interna dell'esponente come un numero esclusivamente positivo, è quella di aggiungere un valore di offset o deviazione; tale valore viene talvolta indicato come *eccesso*. Per questo formato, una deviazione positiva viene aggiunta a tutti gli esponenti, cosicché il numero assume la forma seguente:

$$N.n = fr^{e'} \quad N_b \tag{3.21}$$

dove e' è il valore effettivo dell'esponente memorizzato e N_b è la deviazione positiva. Per un esponente di *L* bit in una base binaria, il numero positivo aggiunto è di solito della forma:

$$N_b = 2^{L-1} \tag{3.22}$$

sebbene il formato standard IEEE discusso nel prossimo sottoparagrafo impieghi un valore $N_b - 1$.

Esempio 3-22

Un formato in virgola mobile dell'IBM ha la seguente rappresentazione per una word di 32 bit:

(a) Il segno come bit più significativo (bit più a sinistra);
(b) I 7 bit successivi come esponente con eccesso 64 o 40_{16} con radice 16;
(c) I 24 bit successivi come frazione in base 16.

Quindi +1.0 è rappresentato come:

$$(0.1)_{16} \times 16^1$$

per cui l'esponente memorizzato diviene 41_{16}. La rappresentazione interna è:

$$4110\ 0000_{16}$$

Esempio 3-23

Un formato in virgola mobile del PDP–11 (della Digital Equipment Corporation) impiega le seguenti convenzioni in una word di 32 bit:

(a) Il segno come bit più significativo (bit più a sinistra);

(b) I successivi 8 bit come esponente nella notazione in eccesso 128 con radice 2;

(c) I successivi 23 bit come frazione. Questi 23 bit sono ricavati da una frazione di 24 bit sempre normalizzata (cioè, il bit più a sinistra sarà 1 in un numero non nullo). Il bit più significativo della frazione normalizzata non viene memorizzato, poiché esso vale sempre 1.

La rappresentazione di +12 è allora:

$$0.11_2 \times 2^{132-128}$$

che è rappresentata internamente come:

$$0\ 1000\ 0100\ 1000\ 0000\ 0000\ 0000\ 0000\ 000_2$$

Si noti che il bit di testa della frazione è un 1 e che esso non è memorizzato col numero in virgola mobile. Questo è stato denominato *bit nascosto* e dovrebbe essere ripristinato da un processore hardware in virgola mobile allorché il valore viene elaborato.

ESERCIZI

3.3.1.1 Si discutano i vari fattori che influenzano la scelta della lunghezza dell'esponente, della lunghezza della mantissa, e la scelta della radice per un numero in virgola mobile. Si calcolino i vari intervalli per una scelta di un esponente di L bit nella notazione di eccesso, una frazione di k bit ed una lunghezza totale di m bit.

3.3.1.2 Si esprima 1/32 in un formato binario in virgola mobile usando un esponente in eccesso 128 di 8 bit ed una frazione di 24 bit con bit di testa implicito. L'ordine nella word da sinistra a destra è segno, esponente e poi frazione (PDP–11).

3.3.1.3 Si convertano i numeri indicati di seguito in una rappresentazione in virgola mobile di 32 bit, con queste caratteristiche: il bit di testa (bit 31) è il segno del numero; i successivi 9 bit sono l'esponente nella notazione di eccesso 256, dopodiché seguono 22 bit di frazione; un numero negativo è rappresentato come il complemento a 2 dell'intero del numero positivo in virgola mobile.
(a) +16.0
(b) –1.0

3.3.2 Formato standard di virgola mobile

Sebbene il processore MC68020 della Motorola non disponga di istruzioni in virgola mobile, molti prodotti della Motorola gestiscono il formato standard di virgola mobile proposto dall'IEEE. Tali prodotti comprendono routine software, routine in memoria a sola lettura, ed un chip di coprocessore che gestisce l'aritmetica in virgola mobile nei sistemi basati sull'MC68020.

Il formato fondamentale consente la rappresentazione di un numero in virgola mobile in un formato singolo o di 32 bit, come segue:

$$N.n = (-1)^S 2^{e'-127}(1.f) \tag{3.23}$$

dove S è il bit di segno, e' è l'esponente deviato (polarizzato), e f è la frazione memorizzata, normalizzata senza l'1 di testa. Internamente, l'esponente ha una lunghezza di 8 bit, mentre la frazione memorizzata è lunga 23 bit. Un formato in doppia precisione consente una rappresentazione di 64 bit, con un esponente di 11 bit ed una frazione di 52 bit.

Varie caratteristiche di questi formati in virgola mobile sono presentate nella Tab. 3.5 e nella Fig. 3.1. Altri formati in precisione estesa sono reperibili nei riferimenti bibliografici relativi a questo capitolo, elencati nell'app. E alla fine del libro. Il coprocessore in virgola mobile MC68881 sarà descritto nel cap. 12.

Tab. 3.5 *Notazione di virgola mobile nello standard IEEE.*

	Singolo	**Doppio**
LUNGHEZZA IN BIT		
Segno	1	1
Esponente	8	11
Frazione	23 + (1)	52 + (1)
Totale	*m* = 32 + 1	*m* = 64 + (1)
ESPONENTE		
Max *e'*	255	2047
Min *e'*	0	0
Deviazione	127	1023

Nota: le frazioni sono sempre normalizzate, e l'1 di testa (bit nascosto) non viene memorizzato.

s	$e + 127$	f
31	30 … 23	22 … 0

← Numero di bit

(a) *Singolo.*

s	$e + 1023$	f
63	62 … 52	51 … 0

(b) *Doppio.*

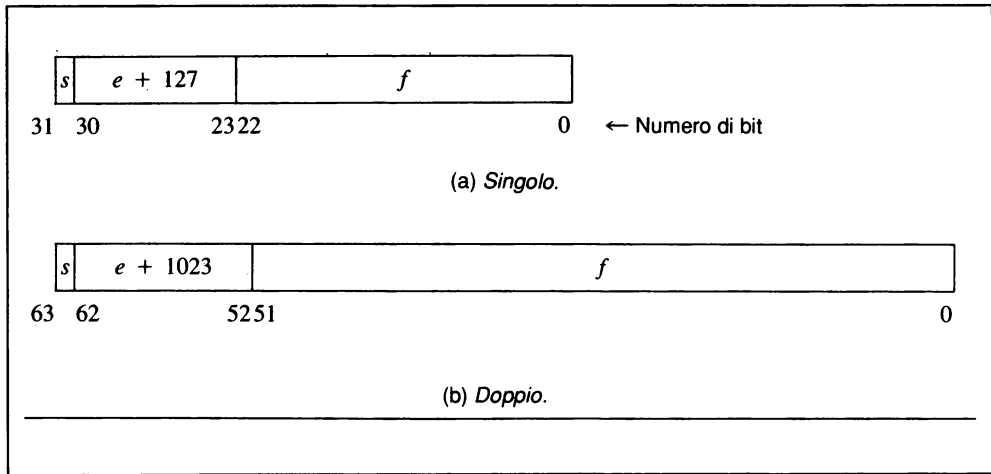



Fig. 3.1 *Formato interno per numero di bit.*

Esempio 3-24

I numeri + 1.0, + 3.0 e −1.0 hanno le seguenti rappresentazioni nel formato standard di 32 bit:

(a) Poiché 1.0 = 1.0 x 2^0, il valore interno è 3F80 0000_{16}.

(b) Poiché 3.0 = 1.5 x 2^1, l'esponente è 128 e la frazione è 1.100_2, senza l'1 di testa. Quindi il valore interno è 4040 0000_{16}.

(c) Poiché −1 = −1.0 x 2^0, il risultato richiede che il bit di segno valga 1 nella rappresentazione BF80 0000_{16}.

ESERCIZI

3.3.2.1 Si scriva la rappresentazione di macchina interna per i seguenti numeri nel formato standard di virgola mobile in doppia precisione:

(a) 0.5
(b) −0.5
(c) 2^{-126}

3.3.2.2 Qual è il valore decimale del massimo numero positivo che può essere rappresentato nel formato standard in singola precisione, se l'esponente polarizzato è limitato ad un massimo di 254 quando viene presentato un numero valido? (Il valore 255 è riservato ad operandi speciali.)

3.3.2.3 Si esprimano i seguenti numeri nella rappresentazione interna, usando il formato standard di virgola mobile in doppia precisione.
(a) 7.0
(b) −30

3.4 RAPPRESENTAZIONE ASCII DI CARATTERI ALFANUMERICI

Poiché con *m* cifre binarie si possono rappresentare 2^m stati distinti, è possibile assegnare un significato ad ogni possibile combinazione delle *m* cifre per produrre un codice che rappresenta informazioni alfabetiche, numeriche, o di altra natura. Uno degli impieghi principali dei codici è quello di consentire al computer di trattare al suo interno i dati di ingresso o di uscita espressi in forma leggibile dagli esseri umani. Tali rappresentazioni binarie interne sono raramente desiderate come uscite, tranne che per eventuali scopi di debugging. Pertanto, la maggior parte dei sistemi di computer disporranno di apposite routine (o di hardware) per convertire i codici binari interni in forma leggibile e viceversa.

Il codice adottato dalla Motorola per rappresentare i caratteri alfanumerici è il codice ASCII (*American Standard Code for Information Interchange*), che è illustrato nell'app. A di questo libro. I codici di 7 bit mostrati nell'appendice sono i valori esadecimali dei caratteri ASCII, così come sarebbero registrati nella memoria. Per esempio, il valore ASCII "1" sarebbe memorizzato come 31_{16}, cioè $0011\ 0001_2$.

I numeri, le lettere ed i caratteri speciali riconosciuti dall'assemblatore e da altri programmi di sistema della Motorola sono memorizzati internamente in codice ASCII come valori di 8 bit, in cui un singolo byte viene usato per memorizzare un singolo carattere. In uscita verso dispositivi quali terminali CRT o stampanti parallele, il codice ASCII viene trasmesso invariato. Se i valori interni sono in formato binario o in BCD, essi devono essere convertiti nel formato ASCII prima di essere inviati in uscita a dispositivi esterni che richiedono la codifica ASCII. I riferimenti bibliografici nell'app. E forniscono un certo numero di esempi di tali conversioni che saranno utili per il programmatore in linguaggio assembler. I programmi per questo scopo saranno presentati nel cap. 7 di questo libro.

È disponibile un certo numero di altri codici per applicazioni di computer, ciascuno con caratteristiche e vantaggi peculiari. Diversi riferimenti blibliografici per questo capitolo discutono i codici in maggiori dettagli. In particolare, Mackenzie fornisce una discussione completa dei codici d'impiego comune ed una presentazione completa del codice ASCII.

Esempio 3-25

La stringa di caratteri ASCII "INPUT" ha la seguente rappresentazione interna:

49 4E 50 55 54

in cui ogni gruppo di due cifre (esadecimali) rappresenta un carattere alfabetico.

ESERCIZI

3.4.1 Quanti byte di memoria sono richiesti per memorizzare il testo di un libro con 100000 parole, se il testo dev'essere memorizzato in codice ASCII? Si supponga che occorrano mediamente 5 caratteri per ogni parola. Se la memoria di un sistema basato sull'MC68020 contiene 2^{24} byte, quale percentuale della memoria viene utilizzata per il testo?

3.4.2 Si converta il testo seguente in codice ASCII (rappresentazione esadecimale interna):"THE MOTOROLA MC68020"

3.4.3 Si esprima la rappresentazione di macchina del numero 255 nei seguenti modi:
(a) Binario
(b) Decimale codificato in binario
(c) ASCII

3.4.4 Si definisca il metodo di conversione e si convertano i seguenti numeri come indicato:
(a) 45 (BCD) a binario
(b) 45 (BCD) ad ASCII
(c) –45 (BCD nella notazione in complemento a 10) ad ASCII, assumendo un numero di 3 cifre con segno.

3.4.5 Si può utilizzare una singola variabile binaria per rappresentare il codice Morse?

CAPITOLO 4

INTRODUZIONE ALL'MC68020

In questo capitolo sono presentate le caratteristiche dell'MC68020 come elemento circuitale e come processore programmabile. Tali caratteristiche sono d'interesse per il progettista di sistemi, il programmatore, ed il progettista d'interfacce. Perlopiù la terminologia usata è conforme a quella impiegata dalla Motorola nella sua letturatura tecnica. Nella Fig. 4.1 è mostrata la pagina di copertina di un documento di 23 pagine che descrive il prodotto MC68020. Molte delle caratteristiche elencate in questo foglio di specifiche tecniche saranno introdotte e discusse in questo capitolo. Negli ultimi capitoli, un certo numero di caratteristiche saranno spiegate più dettagliatamente.

Le istruzioni che il processore esegue sono codificate nella memoria in una forma di linguaggio-macchina. Queste istruzioni potrebbero essere quelle create da un programmatore il quale codifichi direttamente tali sequenze binarie. Più probabilmente, un programma assemblatore convertirà le istruzioni in linguaggio assembler, scritte in notazione simbolica, nelle istruzioni in linguaggio-macchina. Il programmatore in linguaggio assembler di rado lavorerà direttamente col programma in linguaggio-macchina, ma una conoscenza dei formati del linguaggio-macchina è necessaria per una comprensione approfondita delle capacità del processore.

In questo capitolo è presentato l'MC68020 come circuito integrato. Nei prossimi paragrafi saranno descritti l'insieme di registri, l'insieme di istruzioni e le modalità d'indirizzamento, che nel complesso definiscono il modello di programmazione del processore. Sarà presentato anche il linguaggio-macchina dell'MC68020, per illustrare la corrispondenza tra le configurazioni di bit delle istruzioni ed il funzionamento del processore. Il capitolo si concluderà con una presentazione dell'organizzazione della memoria in un sistema basato sull'MC68020.

MOTOROLA
SEMICONDUCTORS
3501 ED BLUESTEIN BLVD AUSTIN, TEXAS 78721

MC68020 Technical Summary

32-BIT VIRTUAL MEMORY MICROPROCESSOR

This document contains both a summary of the MC68020 as well as a detailed set of parametrics. The purpose is twofold to provide an introduction to the MC68020 and support for the sophisticated user. For detailed information on the MC68020 refer to the *MC68020 User's Manual.*

The MC68020 is the first full 32-bit implementation of the M68000 Family of microprocessors from Motorola. Using VLSI technology, the MC68020 is implemented with 32-bit registers and data paths, 32-bit addresses, a rich basic instruction set, and versatile addressing modes. The resources available to the MC68020 user consist of the following:

- Virtual Memory/Machine Support
- Sixteen 32-Bit General-Purpose Data and Address Registers
- Two 32-Bit Supervisor Stack Pointers
- Five Special Purpose Control Registers
- 4 Gigabyte Direct Addressing Range
- 18 Addressing Modes
- Memory Mapped I/O
- Coprocessor Interface
- High Performance On-Chip Instruction Cache
- Operations on Seven Data Types
- Complete Floating-Point Support via MC68881 Coprocessor

FIGURE 1 — FUNCTIONAL SIGNAL GROUPS

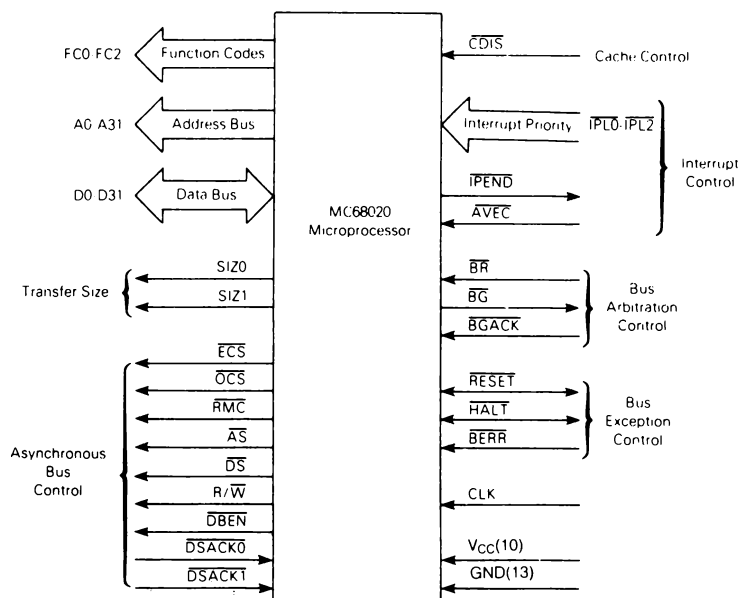



Fig. 4.1 *Sommario tecnico dell'MC68020. (Per gentile concessione di Motorola, Inc.)*

4.1 L'MC68020 COME PROCESSORE A CIRCUITO INTEGRATO

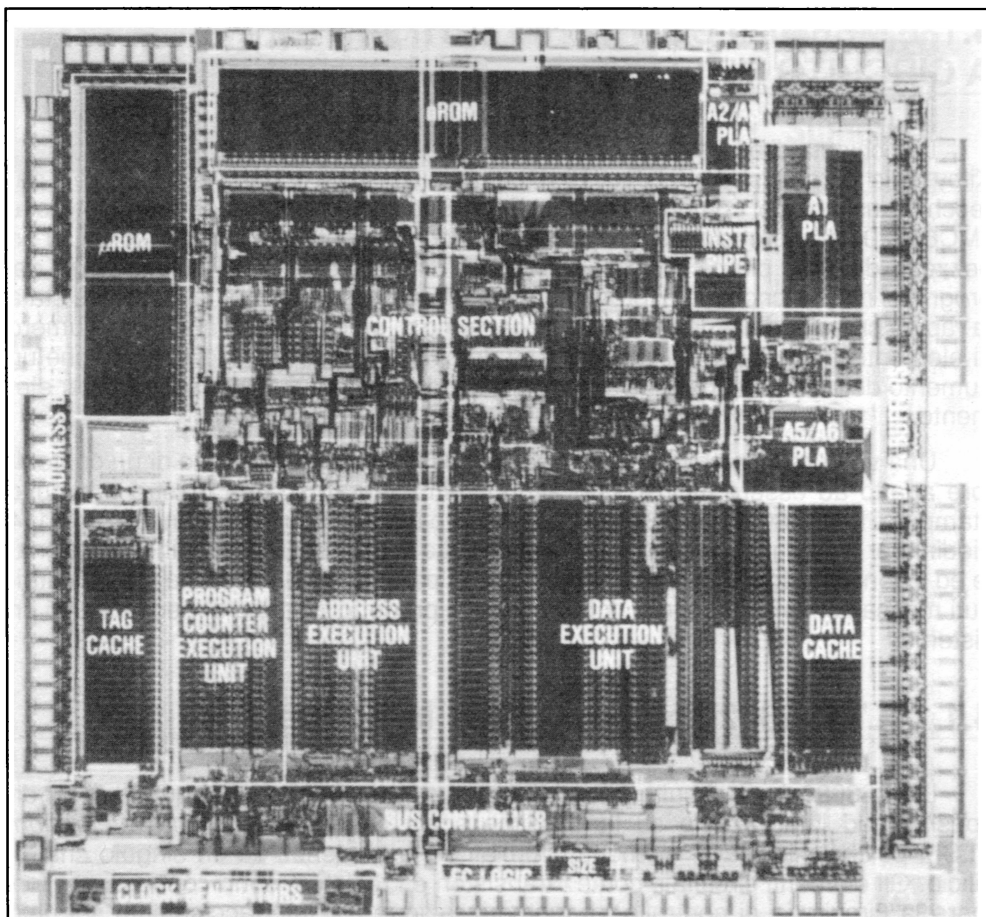
La richiesta di brevetto per uno dei primi circuiti integrati fu presentata da Jack Kilby della Texas Instruments nel 1959 per un dispositivo che era equivalente a parecchi transistori coi componenti associati.[1] Secondo quanto afferma la Motorola, l'MC68020 ha approssimativamente 200000 componenti, che sono integrati su un pezzetto di silicio più piccolo del dispositivo di Kilby. Una delle conseguenze dei progressi nella tecnologia di produzione dei circuiti integrati in oltre 20 anni è stata la rapida crescita della densità d'impaccamento di un chip, misurata come numero di elementi circuitali per unità di area su un singolo chip. Si è registrato anche un aumento della velocità operativa ed una riduzione del consumo di potenza per elemento.

Una volta che un chip è stato prodotto, esso dev'essere dotato di un contenitore adatto ad essere incluso con altri circuiti integrati su una piastra a circuito stampato. L'MC68020, per esempio, è venduto in un contenitore standard con 114 piedini per la connessione al bus di sistema. La funzione di queste linee di segnale ed il consumo totale di potenza elettrica da parte della CPU, come pure gli attributi meccanici del contenitore, influiscono sul progetto elettrico e meccanico di un sistema.

4.1.1 L'MC68020 come circuito integrato

L'MC68020 è un circuito integrato la cui caratteristiche sono quelle di un dispositivo ad integrazione su larghissima scala (*Very Large Scale Integration:* VLSI), poiché migliaia di elementi circuitali sono contenuti su un singolo chip di silicio. Gli elementi circuitali, come i transistori e i resistori, sono connessi per formare la circuiteria di controllo, di memorizzazione e d'interfacciamento sul chip stesso. La Fig. 4.2 illustra l'MC68020 in scala ingrandita prima dell'impaccamento. Il chip effettivo ha dimensioni approssimative di 375 x 350 mil, cioè un'area di circa 130000 mil^2 (1 mil è un millesimo di pollice, pari a circa 25.4 micron). In questa area di neanche 1 cm^2, esso contiene circa 200000 transistori! Le regioni del chip sono illustrate nella Fig. 4.2. Una grande area di ROM sul chip è impiegata per memorizzare il *microcodice*, responsabile dell'esecuzione delle istruzioni in linguaggio-macchina a livello circuitale. Le unità di esecuzione e l'unità aritmetico-logica (*Arithmetic Logic Unit:* ALU) eseguono varie operazioni su indirizzi e dati. Questi valori possono essere trasferiti attraverso la logica del bus di controllo tra la CPU e i dispositivi esterni.

Le istruzioni che causano trappole o salti utilizzano l'appropriata matrice logica programmabile (*Programmable Logic Array:* PLA) per determinare l'ordine in cui le istruzioni saranno eseguite progressivamente dal programma. Nelle PLA sono state implementate delle equazioni logiche per il controllo di altri elementi circuitali entro la CPU. Da questo punto di vista, una PLA è funzionalmente simile alla ROM impiegata per il microcodice.

[1] Jack Kilby e Robert Noyce sono considerati i coinventori del circuito integrato.

LEGENDA:

CONTROL SECTION	= sezione di controllo
INST. PIPE	= pipeline d'istruzione
TAG CACHE	= cache di contrassegno
PROGRAM COUNTER EXECUTION UNIT	= unità di esecuzione del contatore di programma
ADDRESS EXECUTION UNIT	= unità di esecuzione d'indirizzo
DATA EXECUTION UNIT	= unità di esecuzione di dati
DATA CACHE	= cache di dati
BUS CONTROLLER	= controllore di bus
CLOCK GENERATORS	= generatori di clock
FC LOGIC	= logica di FC
SIZE LOGIC	= logica di dimensione
ADDRESS BUFFERS	= buffer d'indirizzo
DATA BUFFERS	= buffer di dati

Fig. 4.2 *Fotografia ingrandita dell'MC68020. (Per gentile concessione di Motorola, Inc.)*

Il metodo di costruzione di un chip è denotato come la sua *tecnologia*. Il nome assegnato ad una particolare tecnologia indica a grandi linee il tipo di transistori impiegati (bipolari o ad effetto di campo), il procedimento di fabbricazione adottato e le caratteristiche del dispositivo finito. I circuiti integrati prodotti con tecnologie diverse sono caratterizzati da differenti densità di elementi per unità di area, differenti velocità operative e differenti consumi di potenza.[2] Per esempio, i circuiti integrati che utilizzano i transistori ad effetto di campo sono denominati dispositivi a metallo-ossido-semiconduttore (MOS), poiché sono fisicamente costruiti in tre regioni costituite da materiali differenti (metallo, un ossido di silicio, ed il substrato di silicio). Vari transistori sono combinati con altri elementi circuitali per formare una cella. Ognuna di tali celle potrebbe contenere, ad esempio, un bit d'informazione. Le aree di ROM del chip rappresentano un gran numero di tali celle, che nell'insieme costituiscono una memoria.

La tecnologia di fabbricazione dell'MC68020 è quella dei transistori a metallo-ossido-semiconduttore complementari ad alta velocità (*High-speed Complementary Metal-Oxide-Semiconductor: HCMOS*). Questa tecnologia consente di ridurre le dimensioni degli elementi rispetto al processo MOS standard e quindi permette di ottenere un aumento di velocità ed una riduzione del consumo di potenza per ciascun transistore nella CPU.

4.1.2 Contenitori, linee di segnale e consumo di potenza

I microprocessori ed altri circuiti integrati prodotti dalla Motorola sono disponibili in un certo numero di diversi contenitori fisici. L'MC68020 a 32 bit con 114 linee di segnale richiede un contenitore del tipo PGA (*Pin-Grid Array:* matrice a griglia di piedini), illustrato nelle Figg. 4.3 e 4.4. Il metodo di incapsulamento determina le dimensioni e la robustezza meccanica del chip completo. Un determinato contenitore influisce anche sulla quantità di calore che può essere dissipata dal chip. Ad ogni modo, la stessa tecnologia impiegata per fabbricare il chip determina il consumo di potenza elettrica della CPU.

Contenitori. Uno degli ultimi passi nella produzione di una CPU a circuito integrato è quello di dotare il dispositivo di un apposito contenitore. Il chip di silicio, mostrato ingrandito in Fig. 4.2, è inserito in un involucro protettivo, il cosiddetto *package* (contenitore), che lo protegge dall'ambiente esterno e ne consente la connessione elettrica ad altri elementi del sistema. Il tipo di contenitore influisce sul costo, sull'intervallo di temperatura di funzionamento, sulle dimensioni, sulla robustezza meccanica e su simili proprietà del prodotto completo. L'MC68020 è incapsulato in un contenitore pin-grid array (PGA) con 13 piedini per lato. Benché l'intera matrice 13 x 13 potrebbe disporre di 169 piedini, soltanto 114 ne sono utilizzati per

[2] I circuiti integrati possono essere fabbricati usando varie altre tecnologie. Oltre al processo HCMOS usato per l'MC68020, sono comuni le tecnologie *bipolare* e a *MOS complementari* (*Complementary MOS: CMOS*).

MC68020
Contenitore con suffisso RC
Dettagli meccanici preliminari

DIM	MILLIMETRI		POLLICI	
	MIN	MAX	MIN	MAX
A	34.18	34.90	1.345	1.375
B	34.18	34.90	1.345	1.375
C	2.67	3.17	.100	.150
D0	.46	.51	.017	.019
G	2.54 BSC		.100 BSC	
K	4.32	4.82	.170	.190
V	1.74	2.28	.065	.095

Vista dall'alto

Piedino A1

Vista dal basso

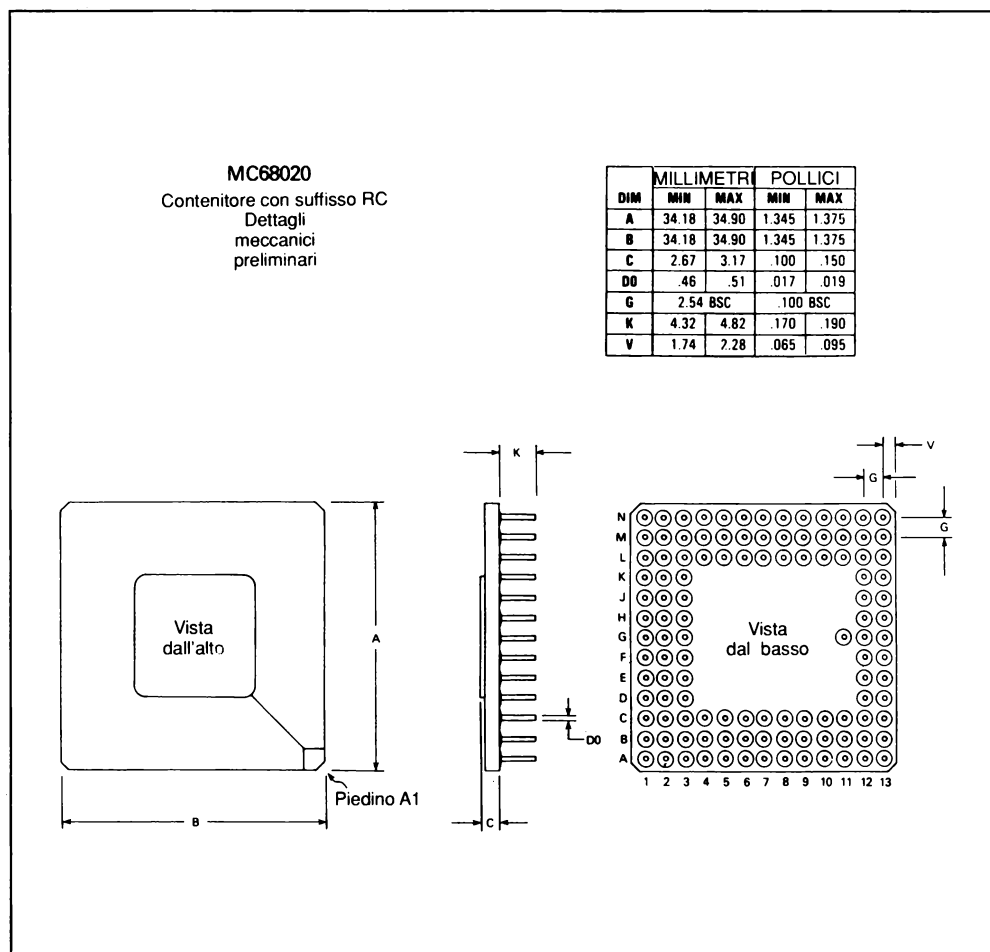
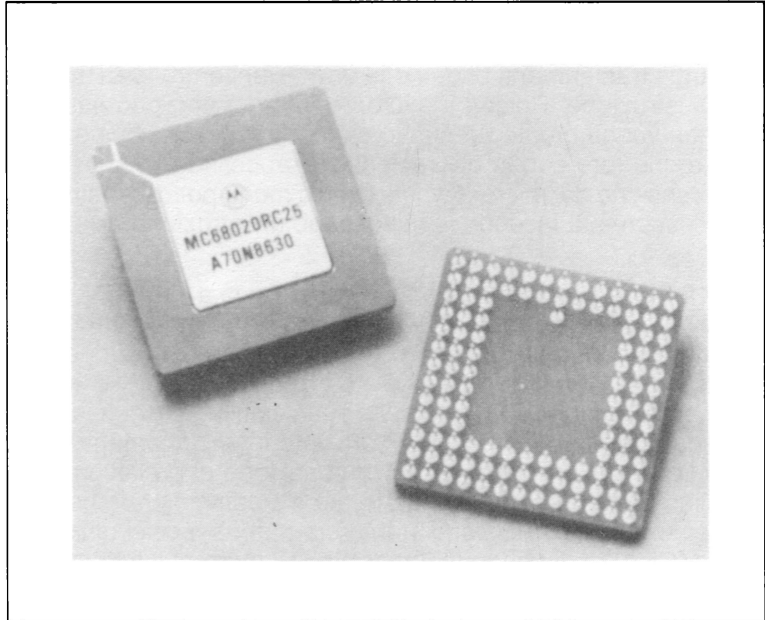



Fig. 4.3 *Dimensioni del contenitore ed assegnazioni dei piedini dell'MC68020. (Per gentile concessione di Motorola, Inc.)*

l'MC68020. La Fig. 4.3 illustra le dimensioni del contenitore e le assegnazioni dei piedini per l'MC68020. La Fig. 4.4 è una fotografia del contenitore dell'MC68020 pronto per essere inserito su una piastra a circuito stampato per la connessione ad altri componenti del sistema.

Linee di segnale per l'MC68020. Un diagramma funzionale delle linee di segnale dell'MC68020 è incluso nella Fig. 4.1. Le 114 linee di segnale controllano l'attività di trasferimento dei dati, le richieste d'interruzione, e operazioni simili. Le linee di segnale hanno origine nel chip di silicio e sono connesse ai piedini del contenitore pin-grid array. I piedini costituiscono quindi la connessione elettrica tra il chip di silicio effettivo dell'MC68020 ed il bus interno di sistema della piastra a circuito stampato su cui il contenitore viene montato.

Fig. 4.4
Contenitore pin-grid array per l'MC68020. (Per gentile concessione di Motorola, Inc.)

Una piastra a circuito stampato completa era stata mostrata nella Fig. 1.2, che illustrava la piastra di processore MVME133. Se s'impiega un metodo di progettazione modulare, le unità di elaborazione, la memoria e le unità d'interfacciamento di un computer consistono di un certo numero di piastre a circuito stampato o di moduli le cui linee di segnale sono interconnesse tramite un bus che è condiviso da ogni modulo del sistema. Un esempio di un bus siffatto è il VMEbus, che sarà descritto nel cap. 14.

Consumo di potenza. La versione dell'MC68020 in un contenitore pin-grid array a 114 piedini ha un consumo di potenza di circa 1.75 watt. In un computer con centinaia di circuiti integrati, il consumo di potenza e conseguentemente il calore generato possono risultare eccessivi. Per esempio, il computer a piastra singola MVME133, descritto nel par. 1.1, richiedeva circa 25 watt. Poiché è un dato di fatto che il riscaldamento eccessivo è una delle principali cause di guasto nei chip, potrebbe essere necessaria una ventilazione forzata anche in un sistema con poche piastre a circuito stampato ed un consumo di potenza evidentemente basso.

4.1.3 Progettazione del processore

Come per molti microprocessori, il funzionamento a livello circuitale dell'MC58020 è controllato da una sequenza *microprogrammata* di istruzioni memorizzate in due sezioni di ROM entro il chip della CPU. Il prelievo delle istruzioni in

linguaggio-macchina dalla memoria esterna e la loro decodifica causa l'esecuzione di un microprogramma. Quest'ultimo controlla tutta l'attività delle linee di segnale e tutti i trasferimenti di dati o le operazioni entro la CPU durante l'esecuzione di quella istruzione. Poiché il microprogramma non può essere modificato, a meno che non venga creato un nuovo chip con un microcodice differente, l'utente ha di rado a che fare con le operazioni del processore a questo livello. Comunque, la comprensione del microcodice è l'unico modo per determinare esattamente ciò che il processore sta facendo in risposta ad un'istruzione.

La Fig. 4.5(a) rappresenta un diagramma a blocchi semplificato delle principali sezioni interne dell'MC68020. Gli elementi mostrati costituiscono un punto di vista funzionale della CPU, anziché una rappresentazione fisica, come quella illustrata in Fig. 4.2. L'MC68020 è stato progettato per il massimo delle prestazioni, così come vengono misurate dalla sua velocità di esecuzione delle istruzioni. Esso impiega una memoria cache e consente il funzionamento in parallelo (concorrente) di varie sezioni entro la CPU. La sezione di prelievo e decodifica contiene un "pipeline" (letteralmente: condotto) a tre stadi, mostrato nella Fig. 4.5(b). Tutti questi elementi della CPU sono stati progettati da tecnici della Motorola per consentire all'MC68020 di eseguire il massimo numero di istruzioni al secondo, usando le tecniche frequentemente impiegate nella progettazione delle moderne unità centrali di elaborazione.

Il cache di istruzioni di 256 byte è stato progettato per ridurre il tempo di esecuzione globale in due modi. Innanzitutto, se un'istruzione si trova in un cache, il prelievo dal cache richiede soltanto due cicli di clock. Sono richiesti tre cicli se l'istruzione dev'essere prelevata dalla memoria principale.[3] In secondo luogo, se l'istruzione viene trovata nel cache, le linee di segnali di dati e di indirizzi sono libere di consentire il trasferimento simultaneo di operandi di dati mentre viene elaborata l'istruzione nel cache. All'interno della CPU, il cache è controllato dal controllore del bus, che coordina e controlla anche tutti gli accessi alla memoria principale. Il controllore del bus può operare indipendentemente dalle altre unità in certi casi e quindi effettua gli accessi alla memoria durante l'esecuzione di un'altra istruzione. Come sarà descritto nel par. 12.1, il cache può essere controllato dal sistema operativo del computer. Per esempio, esso può essere disabilitato (bypassato) dal sistema operativo se le istruzioni devono essere prelevate soltanto dalla memoria principale.

L'unità di prelievo e decodifica dell'istruzione della CPU consiste di un pipeline a tre stadi per la convalida, l'interpretazione e la decodifica delle istruzioni. Il pipeline può contenere tre istruzioni di 16 bit o tre word di 16 bit di una singola istruzione. Come mostrato nella Fig. 4.5(b), il pipeline viene caricato con dati del cache o della memoria principale dal controllore del bus. Quest'ultimo può effettuare un prelievo anticipato (*prefetch*) delle istruzioni dal cache o dalla memoria principale per mantenere pieno il pipeline. Ad ogni stadio, l'istruzione viene ulteriormente decodificata finché, nello stadio D, l'istruzione è stata completamente decodificata.

[3] Nella versione a 16.67 MHz dell'MC68020, ogni ciclo di clock dura 60 nanosecondi. Un prelievo dalla memoria cache richiede quindi 120 nanosecondi.

Sequenziatore
Unità di controllo
Prelievo anticipato (prefetch) e decodifica dell'istruzione
Controllore del bus
Bus di controllo
Bus di indirizzi
32
Unità di esecuzione
Cache di istruzioni
32
Bus di dati

(a)

Prelievo e decodifica dell'istruzione
Sequenziatore
Stadio D
Stadio C
Stadio B
Unità di controllo
Flusso di istruzioni dal cache e dalla memoria
Unità di esecuzione

Nota: Non è mostrato lo stadio A, che contiene le istruzioni prelevate anticipatamente.

Fig. 4.5 *(a) Diagramma a blocchi dell'MC68020. (b) Pipeline. (Per gentile concessione di Motorola, Inc.)*

A quel punto, essa è pronta per essere eseguita. Il tempo di elaborazione globale per una serie di istruzioni viene ridotto col pipeline, poiché l'elaborazione di una nuova istruzione introdotta nel pipeline può iniziare prima che quelle precedenti siano state completate. In effetti, si può ottenere un'elaborazione in parallelo di porzioni di istruzioni diverse.

All'interno del sequenziatore e dell'unità di controllo ci sono due unità ROM, denotate rispettivamente come microROM e nanoROM. Le istruzioni della micro-ROM controllano lo svolgimento in sequenza di micro-operazioni, mentre la nano-ROM controlla l'attività dell'unità di esecuzione. Quest'ultima contiene tre unità artimetico-logiche (ALU) per consentire il calcolo in parallelo di risultati aritmetici, indirizzi di operandi e indirizzi di istruzioni.

L'MC68020 col suo pipeline e la sua architettura interna con un alto grado di parallelismo è in grado di eseguire una nuova istruzione ogni due cicli di clock, se l'istruzione risiede nella memoria cache. Tuttavia, la possibile sovrapposizione dei tempi di esecuzione delle istruzioni e la capacità di prelievo anticipato rendono difficile una previsione delle operazioni del processore ciclo per ciclo, allo scopo di fare stime dei tempi per un programma. Le considerazioni di temporizzazione per l'MC68020 saranno svolte nel par. 13.3.

Esempio 4-1

Secondo quanto affermano i progettisti della Motorola, i processori della famiglia dell'MC68020 impiegano delle CPU microprogrammate per semplificare la progettazione, la modifica e il collaudo della linea di prodotti. Il microcodice a due livelli è stato scelto come un compromesso tra l'efficienza di esecuzione e la dimensione della memoria di controllo (ROM) sul chip della CPU. In questo metodo, ogni istruzione in linguaggio-macchina viene emulata da una sequenza di microistruzioni che, a loro volta, fanno riferimento a "nanoistruzioni" più dettagliate che controllano direttamente l'unità di esecuzione ed altre sezioni della CPU.

Un importante vantaggio dei processori microprogrammati per il produttore è che membri diversi della famiglia possono eseguire la medesima istruzione se il microcodice è comune ai processori della famiglia. L'MC68020 esegue quindi le istruzioni in linguaggio-macchina dell'MC68000 a 16 bit senza alcuna modifica. Naturalmente, l'opposto non è vero, poiché l'MC68020 ha istruzioni che non erano disponibili nell'MC68000. Infine, sono disponibili istruzioni microprogrammate nella ROM dell'MC68020 per il test della CPU. Tali routine di test possono esaminare i percorsi logici ed i circuiti della CPU ad un livello che risulta inaccessibile da parte delle istruzioni in linguaggio-macchina. Tali test sono necessari durante la produzione della CPU, per verificare il corretto funzionamento del chip.

4.1.4 Funzionamento del processore

La funzione dell'architettura interna con pipeline dell'MC68020 è quella di consentire al processore di eseguire le istruzioni ad una velocità maggiore. Nella maggior parte dei casi, la presenza di un pipeline fa aumentare il numero di istruzioni eseguibili in un secondo, rispetto all'esecuzione delle stesse istruzioni senza sovrapposizione nel tempo. Comunque, l'attività del sistema deve svolgersi come se una serie di istruzioni del programma fossero eseguite senza sovrapposizione alcuna. Poiché il programma tipico viene scritto presupponendo che una certa istruzione sia stata completata prima che venga iniziata la successiva, la CPU è progettata per impedire qualsiasi conflitto causato da una sovrapposizione interna. Per tutti i fini pratici, l'MC68020 può essere considerato come se eseguisse una sequenza di istruzioni nell'ordine sequenziale prestabilito. Quindi il programmatore può considerare l'MC68020 come una macchina sequenziale ed ignorare le operazioni del cache e del pipeline. Se l'hardware dev'essere progettato o collaudato ciclo per ciclo, allora può essere necessario prendere in considerazione la sequenza microprogrammata. Per esempio, grazie alla caratteristica di prelievo anticipato (*prefetch*)dell'MC68020, le linee di segnale del processore possono procedere alle operazioni di indirizzamento e lettura di una nuova locazione della memoria principale prima che l'operazione precedente sia terminata. All'occorrenza, la memoria cache può essere disabilitata dal sistema operativo o da un segnale esterno se ciò serve a facilitare il debugging dell'hardware del sistema.

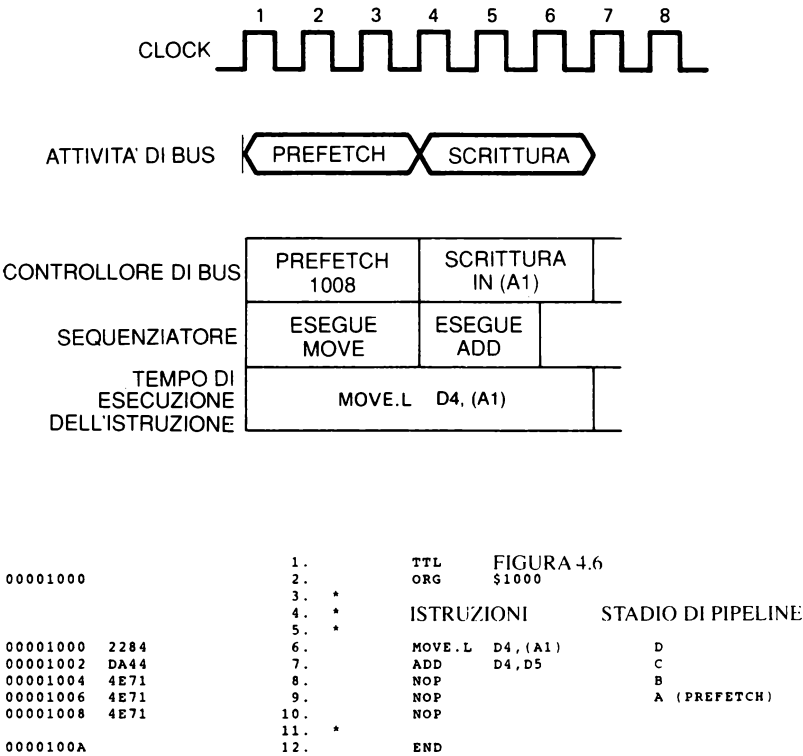
Esempio 4-2

Si supponga che una sequenza di istruzioni debba trasferire i dati da un registro interno di CPU ad una locazione nella memoria e poi sommare i contenuti dei due registri interni. La sequenza di operazioni senza l'impiego della memoria cache è illustrata nella Fig. 4.6. L'istruzione MOVE di 16 bit è contenuta nella locazione 1000, mentre l'istruzione ADD di 16 bit è nella locazione di memoria 1002. Ognuna di queste locazioni contiene il cosiddetto codice operativo (abbreviato in "cod.op.") della relativa istruzione in linguaggio-macchina. Nella figura i codici operativi sono stati espressi in esadecimale. Il pipeline contiene le due istruzioni negli stadi D e C, rispettivamente, prima dell'esecuzione delle istruzioni. Lo stadio B ha il contenuto della locazione 1004. Nell'esempio, l'istruzione è una di "nessuna operazione" (*No OPeration:* NOP). Lo stadio A, impiegato per la memorizzazione temporanea prima che un'istruzione entri nel pipeline, ha il contenuto della locazione 1006.

Nella fase iniziale della sequenza, il controllore del bus effettua un prelievo anticipato del contenuto della locazione 1008 di longword, mentre vengono completate le operazioni interne per l'istruzione MOVE. Ciò richiede tre cicli di clock. L'istruzione ADD (ora nello stadio D) può essere eseguita indipendentemente dal controllore del bus, poiché tale istruzione riguarda soltanto registri interni. Simultaneamente, il valore dei dati contenuti nel registro

numero 4 (D4) sono trasferiti nella memoria in tre cicli di clock dall'istruzione MOVE. Il registro d'indirizzo A1 contiene l'indirizzo della locazione di memoria che rappresenta la destinazione, come indicato nel programma dalla notazione simbolica (A1).

Quando sono viste in termini del numero di cicli di clock all'esterno della CPU, le istruzioni ADD e MOVE richiedono insieme un totale di 6 cicli di clock. Poiché l'istruzione MOVE richiede già di per sé 6 cicli di clock, sembra che l'istruzione ADD sia stata eseguita in zero cicli di clock. Nel ciclo di istruzioni successivo (non mostrato), il processore esegue l'istruzione NOP dalla locazione 1004, mentre vengono decodificate le istruzioni dalle locazioni 1006 e 1008.


```
                        1.           TTL      FIGURA 4.6
00001000                2.           ORG      $1000
                        3.   *
                        4.   *       ISTRUZIONI            STADIO DI PIPELINE
                        5.   *
00001000  2284          6.           MOVE.L   D4,(A1)      D
00001002  DA44          7.           ADD      D4,D5        C
00001004  4E71          8.           NOP                   B
00001006  4E71          9.           NOP                   A (PREFETCH)
00001008  4E71         10.           NOP
                       11.   *
0000100A               12.           END
```

Note:

(1) D4 e D5 sono le designazioni simboliche per due registri di processore che contengono operandi. A1 è un registro che contiene l'indirizzo della locazione di destinazione nella memoria.

(2) La memoria cache è disabilitata nell'esempio. Pertanto, le istruzioni sono prelevate dalla memoria principale e inserite nel pipeline.

Fig. 4.6 *Sovrapposizione delle istruzioni nell'MC68020.*

ESERCIZI

Alle prime quattro domande si può rispondere consultando i vari articoli o libri di testo elencati nei riferimenti bibliografici relativi a questo capitolo, elencati nell'app. E alla fine del libro.

4.1.1 I membri della famiglia di microprocessori dell'MC68020 sono microprogrammati. Si confronti il progetto di una CPU che usa un controllo microprogrammato con un progetto che utilizza una circuiteria logica ad-hoc (cablata). Si considerino le fasi iniziali di progettazione e test, come pure la velocità operativa e la flessibilità quando è necessario modificare il progetto.

4.1.2 L'MC68020 è considerato un processore CISC (*Complex Instruction Set Computer:* computer con insieme d'istruzioni complesso). Ciò implica che le istruzioni contenute nell'insieme sono relativamente complesse. Un altro metodo di progettazione di una CPU impiega i princìpi dell'architettura RISC (*Reduced Instruction Set Computer:* computer con insieme di istruzioni ridotto). In questo caso le istruzioni risultano più semplici e forse più efficienti. Consultando la letteratura tecnica appropriata, si confrontino i due metodi di progettazione di una CPU.

4.1.3 L'MC68020 impiega sia ROM che PLA (*Programmable Logic Array:* matrice logica programmabile) nella propria CPU per l'ordinamento in sequenza ed il controllo. Benché una PLA sia funzionalmente simile ad una ROM, essa presenta tuttavia delle peculiarità; si discutano le caratteristiche ed i vantaggi di ciascun metodo adottato nella progettazione di una CPU. Questo problema è rivolto a quei lettori che hanno conoscenze di progettazione di circuiti elettronici.

4.1.4 Si discutano i fattori che influiscono sulla scelta della lunghezza del pipeline per l'MC68020. Un pipeline a più stadi dovrebbe far aumentare la velocità di esecuzione, ma si consideri ciò che accade quando la sequenza delle istruzioni viene modificata, per esempio in seguito ad un'istruzione di salto.

4.1.5 Si determini la riduzione percentuale di tempo ed il miglioramento effettivo (in nanosecondi) quando N istruzioni in sequenza sono reperite nella memoria cache (che richiede due cicli di clock per accesso), rispetto al caso in cui esse devono essere prelevate dalla memoria principale (tre cicli di clock per accesso) per l'MC68020 con le seguenti frequenze di clock:
(a) 16,67 MHz
(b) 20 MHz
Questo miglioramento si ha con la memoria cache soltanto per il prelievo delle istruzioni. L'esecuzione di ciascuna istruzione può richiedere un certo numero di cicli di clock in più, oltre al tempo per prelevare l'istruzione.

4.2 IL MODELLO DI PROGRAMMAZIONE DI REGISTRI DELL'MC68020

Proprio come la memoria è usata per immagazzinare le istruzioni e i dati relativi ad un programma, la CPU contiene elementi di memorizzazione denominati *registri*, che contengono le informazioni necessarie per l'istruzione in corso di elaborazione. Un registro consiste di una o più celle di memoria, ciascuna contenente un bit d'informazione. La *lunghezza* del registro è definita come il numero di bit

che possono essere memorizzati o letti simultaneamente. Una CPU contiene un gran numero di registri, la maggior parte dei quali sono utilizzati per scopi specifici entro la CPU. Alcuni di tali registri, definiti registri *programmabili*, sono disponibili al programmatore in linguaggio-macchina o in linguaggio assembler tramite l'insieme di istruzioni del processore.

Come descritto nel cap. 2, l'MC68020 consente di suddividere i programmi a seconda che siano eseguiti nella modalità di utente o in quella di supervisore.[4] Le sue routine hanno un privilegio maggiore di quello di un programma in modo utente. Queste routine di supervisore possono eseguire istruzioni che sono proibite al modo di utente. I programmi nella modalità di utente sono anche impossibilitati a modificare i contenuti di certi registri del processore, quali i puntatori di stack di supervisore ed il registro di stato della CPU. Un programma in modo supervisore può utilizzare qualsiasi registro del processore durante la sua esecuzione.

Questo paragrafo descrive dapprima tutti i registri del processore che sono impiegati per la programmazione di finalità generale. Si tratta dei registri di dati, di sette dei dieci registri d'indirizzo, del contatore di programma e del registro dei codici di condizione. Dopo la loro descrizione, saranno discussi i puntatori di stack di sistema ed il registro di stato. I programmi operanti sia in modo di utente che in modo di supervisore usano i medesimi registri generali.

4.2.1 L'insieme di registri generali dell'MC68020

I registri generali (*general purpose:* di finalità generale) di un processore contengono indirizzi o valori di dati soggetti ad essere trattati da un'istruzione. Il *contatore di programma* contiene l'indirizzo dell'istruzione successiva da prelevare dalla memoria. Il *registro dei codici di condizione* contiene dei bit che indicano i risultati di operazioni aritmetiche o simili. Nell'MC68020, i registri generali sono suddivisi in otto *registri di dati* e sette *registri d'indirizzo*. Questa CPU ha un contatore di programma di 32 bit ed un registro dei codici di condizione di 8 bit.

La Fig. 4.7 illustra l'insieme di registri dell'MC68020 e un diagramma semplificato dei percorsi di trasferimento interni ed esterni per il processore. Le linee dei segnali d'indirizzo, di dati e di controllo sono connesse al bus di sistema come spiegato nel cap. 2. Al suo interno, il processore contiene dei registri programmabili che sono trattati come dati o indirizzi. Questi valori possono essere trasferiti all'unità aritmetico-logica (ALU) per i calcoli aritmetici. I valori dei dati contenuti nei registri di dati del processore possono essere trasferiti da e verso la memoria o tra la CPU e i dispositivi periferici tramite le linee di segnali di dati. I valori nei registri d'indirizzo possono essere posti sul bus indirizzi per far riferimento alle locazioni nella memoria. Queste operazioni sono controllate da istruzioni in linguaggio-macchina,

[4] Nella maggior parte dei computer, soltanto le routine d'importanza critica del sistema operativo sono eseguite nel modo di supervisore. Di solito, si tratta di routine che controllano il funzionamento dell'hardware in qualche maniera. Molti programmi associati col sistema operativo sono registrati in un'unità a disco e caricati nella memoria all'occorrenza. Queste distinzioni entro il sistema operativo non sono importanti per questo paragrafo.

che vengono prelevate dalla memoria tramite le linee di segnali di dati e successivamente decodificate dalla CPU.

REGISTRI DI DATI
31 15 0
D0
D7
REGISTRI DI INDIRIZZO
31 15 0
A0
A6
PUNTAT. DI STACK DI UTENTE
PUNTAT. DI STACK DI SUPERVISORE
CONTAT. DI PROGR.
REGISTRI SPECIALI
DATI 32
32 INDIRIZZO
15 7 0
REGIST. DI STATO
CCR
ALU
CONTROLLO
CONTROL. E TEMPORIZZAZIONE
CLOCK
ALIMENTAZIONE

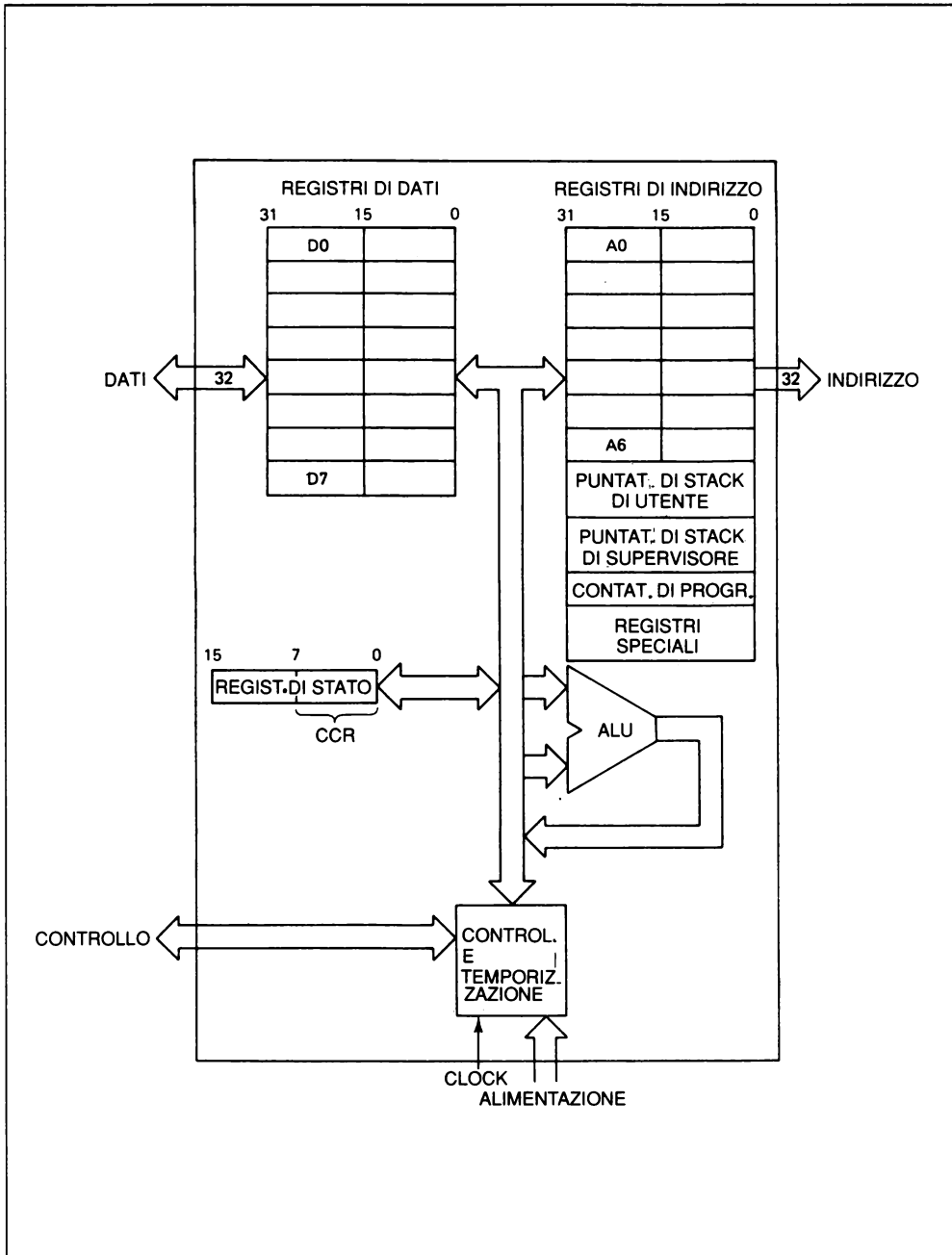



Fig. 4.7 *L'insieme di registri ed i percorsi di trasferimento dell'MC68020.*

L'insieme di registri programmabili di qualsiasi processore può essere definito in termini delle caratteristiche utili ad un programmatore. Il numero, il tipo e la lunghezza (in bit) dei registri e la loro interconnessione attraverso i percorsi di dati interni ed esterni determinano la capacità fondamentale del processore di eseguire le istruzioni mediante questi registri. In generale, se si dispone di un maggior numero di registri, la programmazione risulterà semplificata al livello di linguaggio assembler. Inoltre, la velocità di esecuzione del programma viene aumentata se i registri sono utilizzati per contenere operandi, poiché le operazioni tra registri richiedono meno tempo di elaborazione rispetto alle operazioni che fanno riferimento a dispositivi esterni o alla memoria.

I registri servono anche a contenere gli indirizzi di elementi di dati memorizzati all'esterno del processore. Questi indirizzi possono essere facilmente modificati durante l'esecuzione del programma, modificando il contenuto dei registri. Le moderne tecniche di programmazione favoriscono questo metodo d'indirizzamento

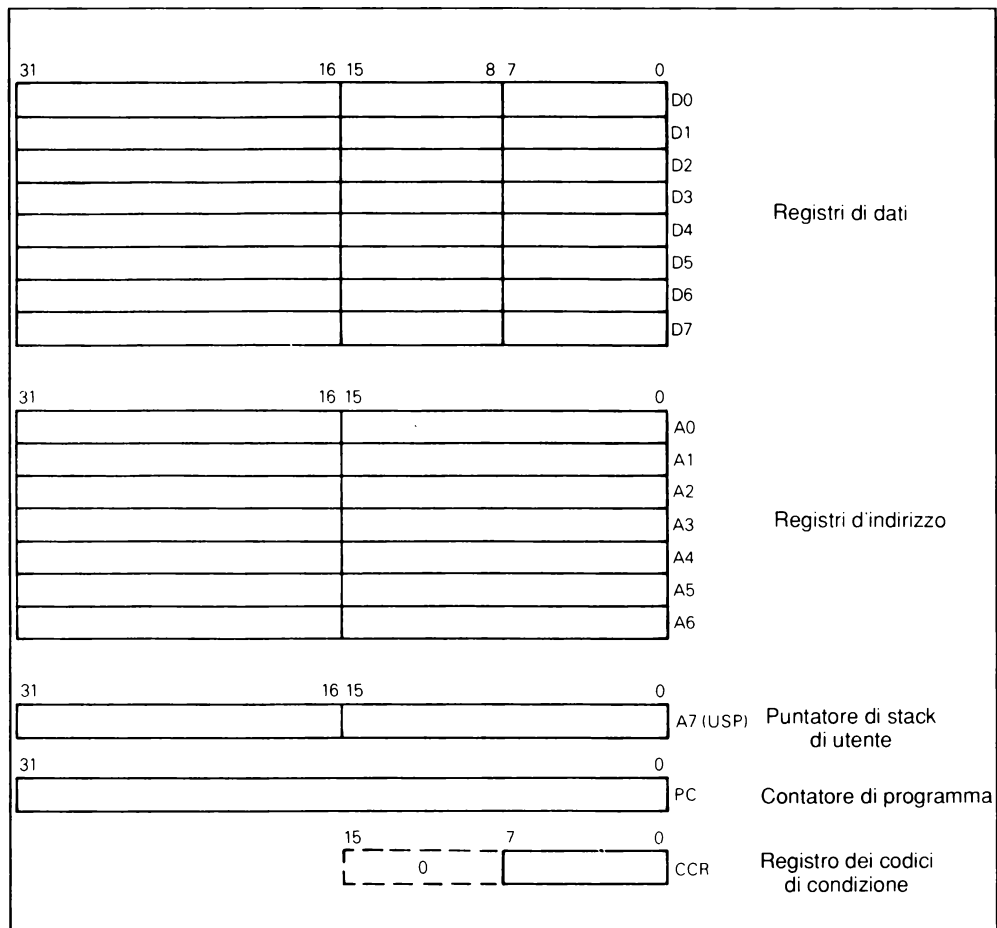



Fig. 4.8 *L'insieme di registri programmabili dell'MC68020 nella modalità di utente.*

indiretto dei registri, invece di consentire al programma di modificare sé stesso cambiando un indirizzo che fa parte di un'istruzione. In molti processori, come nell'MC68020, i registri che contengono l'indirizzo di un operando possono essere modificati dall'aggiunta (o dalla sottrazione) del contenuto di altri registri denominati *registri indice.*

In Fig. 4.8 è mostrato l'insieme specifico di registri disponibili ai programmi di utente di un computer basato sull'MC68020. Oltre ai registri generali appena definiti, tale insieme contiene un puntatore di stack di utente, designato come USP (*User Stack Pointer*) o A7 nella figura.[5]

Lo stack di utente è un'area della memoria utilizzata dalla CPU per risparmiare indirizzi di ritorno durante operazioni quali le chiamate di subroutine. È appunto l'User Stack Pointer che indirizza locazioni specifiche nella memoria per questo stack. La notazione simbolica è impiegata in questo testo e in un programma in linguaggio assembler per ciascun registro della CPU, come mostrato nella Tab. 4.1. Per esempio, il programmatore in linguaggio assembler userebbe tale notazione per designare un registro in un'istruzione del linguaggio assembler. Per far riferimento al puntatore di stack di utente, la notazione A7 o SP sarebbe utilizzata da un programma in modo utente. Questo puntatore di stack ha le medesime caratteristiche degli altri sette registri d'indirizzo, ma è riservato come *puntatore di stack di sistema* quando la CPU opera nel modo di utente. La designazione SP (*Stack Pointer:* puntatore di stack) è impiegata anche per il puntatore di stack di sistema. La designazione USP è riservata ai programmi che vengono eseguiti nella modalità di supervisore della CPU, per motivi che saranno spiegati in seguito.

Il contenuto di un registro o di una locazione di memoria viene designato qui racchiudendo tra parentesi l'elemento. Così (D2) indica il contenuto del registro di dati D2. Quando si vogliono indicare alcuni bit selezionati di un operando, i numeri dei bit sono racchiusi tra parentesi quadre, in cui il numero del bit iniziale e del bit finale sono separati dal segno di due punti (:) se sono specificati bit consecutivi. Per esempio, i bit da 0 a 7 del registro di dati D1 sono indicati da (D1)[7:0]. Un operando designato da <operando> significa che qualsiasi operando valido, come determinato nella discussione, può essere sostituito nell'istruzione. La designazione < Dn> , per esempio, significa si può specificare qualsiasi registro di dati (cioè, uno qualsiasi di D0, D1, ..., D7).

4.2.2 I registri di dati

L'MC68020 ha otto registri designati come registri di dati. Essi vengono denotati simbolicamente tramite il numero, come Dn, in cui n = 0, 1, ..., 7. La struttura del bus interno della CPU consente il trattamento di un'operando di byte (Dn) [7:0], un operando di word (Dn) [15:0], o un operando di longword (Dn) [31:0] nel registro

[5] Un programma in linguaggio assembler nel modo utente fa riferimento al puntatore di stack di utente tramite il codice mnemonico SP o A7. Un programma in modo supervisore fa riferimento al puntatore di stack di utente mediante la designazione USP in un'istruzione del linguaggio assembler. Queste distinzioni saranno ulteriormente discusse nel cap. 5.

Tab. 4.1 *Impieghi e notazioni simboliche dei registri (modalità di utente).*

Registro	Notazione simbolica	Impiego
Dati	D0 D1 . . D7	Accumulatore Registro buffer Registro indice Memoria temporanea
Indirizzo	A0 A1 . . A6	Indrizzamento indiretto Puntatore di stack Registro indice
Puntatore di stack di sistema	A7 o SP	
SP di utente		Chiamate di subroutine (modo di utente)
Contatore di programma	PC	Indirizzamento di istruzione
Registro dei codici di condizione	CCR	Codici di condizione

di dati selezionato. Poiché sono possibili tre lunghezze, le istruzioni del processore che fanno riferimento ad un registro di dati devono indicare la lunghezza dell'operando. Soltanto i bit corrispondenti del registro specificato vengono modificati da quella istruzione. La porzione del registro interessato può essere usata come un accumulatore, o un registro di memorizzazione, o un registro di buffer o come un registro indice.

Come accumulatori, i registri di dati contengono operandi della lunghezza specificata e consentono operazioni aritmetiche, logiche e di altra natura. Questi registri sono usati anche per memorizzare temporaneamente gli operandi generati in altri registri del processore. I registri di dati agiscono come registri di buffer quando i valori di dati sono trasferiti dentro o fuori del processore tramite le linee di segnali di dati. Per l'MC68020 con le sue 32 linee di segnali di dati, i trasferimenti possibili comprendono un valore di 8 bit, o di 16 bit o di 32 bit in un singolo trasferimento.

Un registro di dati può essere usato come registro indice il cui contenuto va sommato al valore presente in un registro d'indirizzo per formare l'indirizzo di un

operando. La potenza di questa capacità d'indirizzamento è tale che l'indice può essere modificato da qualsiasi istruzione del processore che operi su un registro di dati. Il valore dell'indice può essere modificato in modi molto "sofisticati" durante l'esecuzione del programma. Ciò di solito avviene entro un ciclo del programma.

Esempio 4-3

La Fig. 4.9 mostra un registro di dati dell'MC68020, in cui i bit sono designati da 0 (il più a destra) fino a 31 (il più a sinistra). Il bit di segno di un numero in complemento a 2 lungo un byte dovrebbe essere il bit 7, come indicato. Se si specifica un operando di un byte in un accesso qualsiasi a tale registro, allora sarebbero interessati soltanto i bit da 0 a 7; i restanti bit rimarrebbero inalterati. Similmente, il bit di segno di un operando di word è il bit 15, mentre quello di un operando di longword è il bit 31, come mostrato nella figura.

LONGWORD 32 bit
WORD 16 bit
BYTE 8 bit
31 16 15 8 7 0
Bit di segno di byte
Bit di segno di word
Bit di segno di longword

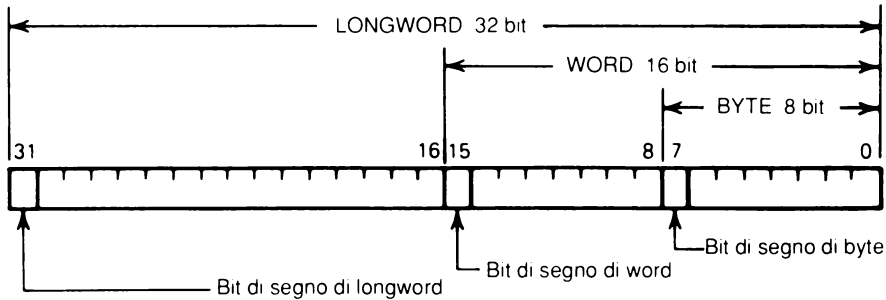



Fig. 4.9 *Formato dei registri di dati.*

4.2.3 I registri d'indirizzo

L'MC68020 ha sette registri d'indirizzo di tipo generale, che accettano soltanto valori di word o di longword. Questi sette registri d'indirizzo, designati simbolicamente come A0, A1, ..., A6, sono condivisi da programmi nel modo di supervisore o nel modo di utente. I sette registri disponibili ad un programma possono essere utilizzati per indirizzare indirettamente gli operandi nella memoria, oppure possono essere impiegati per indirizzare stack "privati" definiti dal programmatore.

L'impiego principale dei registri d'indirizzo è quello di contenere l'indirizzo di un operando nella memoria. Poiché un registro d'indirizzo è lungo 32 bit, l'intervallo possibile per un indirizzo si estende fino a

$$2^{32} = 4294967296$$

locazioni. Nell'MC68020 il contenuto di 32 bit di un registro d'indirizzo può essere inviato in uscita tramite le linee di segnali d'indirizzo del processore.

Stack privati. Nell'MC68020, uno *stack* (pila) consiste di un insieme di locazioni di memoria contigue indirizzate da un registro designato come *puntatore di stack*. Gli elementi memorizzati sullo stack saranno reperiti nell'ordine inverso, in modo simile a quanto avviene per una pila di piatti in un ristorante. L'accesso allo stack, per registrare o reperire i dati da una singola estremità, avviene in maniera LIFO (*Last In First Out:* ultimo dentro, primo fuori).

Da un punto di vista funzionale, un puntatore di stack contiene un valore, designato (SP), utilizzato come un indirizzo per puntare alla cima dello stack. Il processore impiega il valore nel puntatore di stack per indirizzare una locazione da cui il processore legge o in cui scrive un elemento di dati. L'informazione viene inserita o registrata sullo stack, con una cosiddetta operazione di *push*, da un ciclo di scrittura del processore. L'elemento d'informazione viene reperito da un ciclo di lettura del processore, mediante un'operazione di prelievo nota come *pop* o *pull*.

Ogni registro d'indirizzo di finalità generale dell'MC68020 può essere utilizzato come un puntatore di stack privato da un programma che impiega una delle modalità d'indirizzamento appropriate per le operazioni di stack. In queste modalità d'indirizzamento, il valore del registro d'indirizzo da usare come puntatore di stack viene modificato automaticamente (incrementato o decrementato) della quantità appropriata dopo ogni operazione di inserimento o prelievo di un elemento di dati. I valori dei dati nello stack possono essere lunghi un byte, una word o una longword, se uno qualsiasi dei registri designati come A0, A1, ..., A6 sono utilizzati come puntatori di stack.

Registri indice. Un altro impiego di un registro d'indirizzo è quello di registro indice. Il valore dell'indice viene aggiunto al contenuto di un altro registro d'indirizzo per calcolare l'indirizzo effettivo di un operando nella memoria. Il medesimo impiego è stato definito per un registro di dati nel paragrafo precedente. Quindi l'MC68020 permette di utilizzare come registri indice sia i registri d'indirizzo che quelli di dati.

Esempio 4-4

La Fig. 4.10(a) illustra il formato di un registro d'indirizzo dell'MC68020. L'indirizzo può essere lungo 16 bit o 32 bit. Comunque, il corrispondente indirizzo di longword sarebbe lungo 32 bit quando viene emesso sulle linee di segnali d'indirizzo.

La Fig. 4.10(b) mostra un riferimento indiretto alla memoria. In questo esempio, l'indirizzo contenuto in A1 punta ad un operando nella locazione decimale 1000. Dunque l'indirizzo effettivo dell'operando è designato come il contenuto di A1 o (A1).

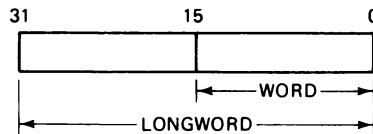
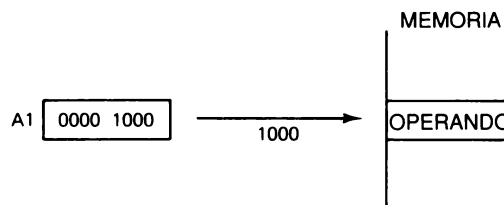
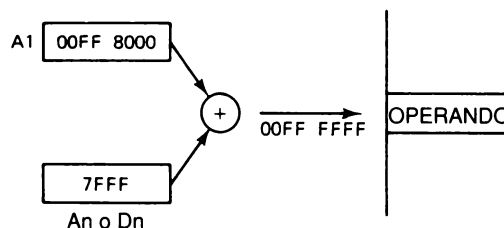
L'indirizzamento indicizzato, mostrato nella parte (c) della figura, consente di sommare due valori da utilizzare per determinare la locazione dell'operando. L'indirizzo esadecimale massimo di 32 bit è FFFF FFFF.

31 15 0
WORD
LONGWORD

(a) *Registro d'indirizzo.*

MEMORIA
A1 0000 1000
1000
OPERANDO

(b) *Riferimento di memoria indiretto.*

A1 00FF 8000
+
00FF FFFF
OPERANDO
7FFF
An o Dn

(c) *Riferimento di memoria indicizzato.*

Nota: Tutti i valori sono in esadecimale.

Fig. 4.10 *Utilizzatore dei registri d'indirizzo dell'MC68020*

Esempio 4-5

Il *fondo* dello stack è il primo elemento inserito nello stack, mentre la *cima* è l'ultimo elemento registrato. La rimozione o prelievo (pop) di un elemento avviene dalla cima. Uno stack privato può crescere da indirizzi inferiori verso indirizzi superiori nella memoria, oppure può crescere verso il "basso" nella memoria. Questi due casi sono illustrati nella Fig. 4.11(a), in cui è anche mostrato l'indirizzamento richiesto. La notazione impiegata qui ((SP')) denota il contenuto dello stack (cioè, il contenuto della locazione indirizzata dal puntatore di stack).

(i) Stack che cresce verso il basso nella memoria

(libera)
(SP) → CIMA
FONDO
Indirizzi inferiori

PUSH: (SP') ← (SP') – k
allora
((SP')) ← Operando

POP: Trasferisce operando da (SP)
allora
(SP')← (SP') + k

(ii) Stack che cresce verso l'alto nella memoria

FONDO
CIMA
(SP) → (libera)
Indirizzi superiori

PUSH: Trasferisce operando a (SP)
allora
(SP') ← (SP') + k

POP: (SP') ← (SP') – k
allora
Trasferisce operando da (SP')

(a) *Stack nella memoria.*

(i) Prima di PUSH

(SP') = 1000 FFFF

(ii) Dopo PUSH

(SP') = 0FFE 1234
1000 FFFF
Indirizzi superiori

(b) *Indirizzamento di stack per una PUSH di un operando di lunghezza di word.*

Fig. 4.11 *Operazioni di stack.*

Quando lo stack cresce verso il basso nella memoria verso indirizzi inferiori, (SP') punta all'ultimo elemento (cima) e dev'essere decrementato prima di un'inserimento (push). Dopo un prelievo, (SP') dev'essere incrementato per puntare di nuovo alla cima. Il procedimento opposto avviene per uno stack che cresce verso l'alto nella memoria. Gli incrementi (o decrementi) sono di 1, 2 o 4, a seconda che la dimensione di ciascun elemento nello stack sia di un byte, di una word o di una longword. Se un'istruzione specifica la modalità di *predecremento* dell'MC68020, la CPU sottrae automaticamente k (con k = 1, 2 o 4) da (SP'), prima che il puntatore di stack sia usato come indirizzo. La modalità di *postincremento* è usata per aggiungere k a (SP') dopo l'uso. Le suddette modalità d'indirizzamento saranno discusse in maggior dettaglio nel par. 4.4.

La Fig. 4.11(b) mostra il contenuto dello stack prima e dopo l'inserimento del valore esadecimale 1234 su uno stack di word (16 bit). Prima dell'inserimento, il puntatore di stack contiene l'indirizzo esadecimale 1000 nell'esempio, ed il valore di ((SP')) è FFFF. Per inserire tale valore, il puntatore di stack viene dapprima decrementato di 2 e poi usato come un indirizzo della locazione di stack per il valore del dato. Pertanto, il valore sarà inserito nella locazione 0FFE.

In quest'esempio, la notazione SP' è utilizzata quando si fa riferimento al puntatore di stack soltanto per questi stack privati. In un'istruzione in linguaggio assembler, dev'essere designato esplicitamente uno dei registri d'indirizzo A0, A1, ..., A6, come sarà spiegato nel cap. 5.

4.2.4 Il contatore di programma

Nella famiglia di processori dell'MC68020, il contatore di programma ha una lunghezza interna di 32 bit. Ciò consente d'indirizzare oltre due miliardi di word nella memoria. Poiché l'MC68020 può avere istruzioni che variano da una word fino a 11 word nella memoria, il contatore di programma viene incrementato automaticamente della quantità appropriata mentre viene eseguita l'istruzione corrente.

Il contatore di programma può essere modificato dal programmatore per variare la sequenza di controllo in vari modi. La normale sequenza di esecuzione può essere modificata direttamente da un'istruzione di programma che causa un salto nel programma stesso. In questo caso, nessun indirizzo di ritorno viene salvato dalla CPU, poiché il controllo non è restituito all'istruzione che segue quella di salto. L'istruzione di salto carica quindi il nuovo indirizzo nel contatore di programma, distruggendone il contenuto precedente. Per contro, una chiamata ad una subroutine o ad un'eccezione causa il salvataggio nello stack di sistema del valore del contatore di programma, prima che il suo contenuto venga modificato dall'indirizzo della nuova routine da eseguire. L'ultima istruzione in una subroutine dev'essere una di quelle — ad esempio, Return — che ripristinano il contenuto del contatore di programma.

4.2.5 Il registro dei codici di condizione

I codici di condizione sono variabili di un solo bit, che indicano i risultati di operazioni logiche o aritmetiche. I loro valori sono assegnati automaticamente da molte istruzioni dell'MC68020. Per esempio, se un'addizione produce una somma nulla, il bit Z viene posto a {1}. Gli altri bit hanno significati analoghi, come mostrato nella Tab. 4.2.

Tab. 4.2 *Interpretazione dei codici di condizione.*

NOME	NUMERO DI BIT	SIMBOLO	SIGNIFICATO
Extend (Estensione)	4	X	Usato nelle operazioni aritmetiche in precisione multipla; in molte istruzioni, esso viene eguagliato al bit C.
Negative (Negativo)	3	N	Posto a {1} se il bit più significativo di un operando vale {1}.
Zero (Zero)	2	Z	Posto a {1} se tutti i bit di un operando sono {0}.
Overflow (Superamento)	1	V	Posto a {1} se si verifica una condizione di fuori-intervallo in un'operazione in complemento a 2.
Carry (Riporto)	0	C	Posto a {1} se viene generato un riporto dal bit più significativo della somma in un'addizione. Posto a {1} se viene generato un riporto negativo in una sottrazione.

Note:
1. I restanti bit [7:5] del registro dei codici di condizione (CCR) non sono utilizzati, ma il CCR è considerato un registro della lunghezza di un byte.
2. Nel modo di supervisore, il CCR è il byte meno significativo del registro di stato.

4.2.6 I puntatori dello stack di sistema

L'MC68020 utilizza gli stack per memorizzare le informazioni quando una chiamata di subroutine viene effettuata da un programma o quando si presenta una *eccezione* durante l'attività del sistema. Queste eccezioni, così come vengono definite dalla Motorola per l'MC68020, includono trappole, interruzioni e diverse condizioni di errore riconosciute dalla CPU. Quando si verifica uno qualsiasi di questi eventi, viene modificato il normale flusso di controllo attraverso le istruzioni sequenziali nella memoria. Il controllo viene trasferito alle istruzioni associate con la subroutine, la trappola o l'interruzione finché non è stato portato a termine il suo compito specifico. Poi il controllo viene restituito normalmente all'istruzione successiva nella sequenza già interrotta. Per consentire la restituzione del controllo, la CPU memorizza o salva le informazioni nello *stack di sistema* appropriato. Per una chiamata di subroutine, soltanto il valore di (PC) dev'essere salvato e ripristinato al termine della subroutine. Quando si presenta un'eccezione, i contenuti del

PC e del registro di stato (*Status Register:* SR) sono salvati, insieme con altre informazioni, conformemente ai requisiti dell'eccezione. Le operazioni di salvataggio e di ripristino sono automatiche e non richiedono alcun intervento da parte del programmatore.

L'MC68020 suddivide l'area assegnata allo stack nella memoria in un'area di stack di utente e in un'area di stack di supervisore. Nella modalità di utente, lo stack di sistema è quello di utente, per cui viene usato il puntatore di stack di utente per indirizzare tale stack. La locazione iniziale di questo stack di utente, prima che esso sia utilizzato per il salvataggio ed il ripristino delle informazioni, viene assegnata dal sistema operativo. Quando la CPU opera nella modalità di supervisore, lo stack di sistema è uno stack di supervisore. Il puntatore di stack di supervisore (*Supervisor Stack Pointer:* SSP) indirizza le aree di stack di supervisore nella memoria. Un programma in modo utente può modificare il puntatore dello stack di utente, ma non può accedere al puntatore di stack di supervisore. Pertanto, anche se un programma in modo utente gestisse scorrettamente il proprio stack e causasse un errore grave, lo stack di supervisore non sarebbe alterato. Il sistema operativo deve trattare correttamente soltanto il proprio stack al fine di garantire un funzionamento corretto del sistema. Si dovrebbe altresì notare che i contenuti dei registri generali di dati o d'indirizzo non vengono salvati automaticamente nello stack di sistema in risposta ad un'eccezione. Se necessario, devono essere usate delle opportune istruzioni del programma per salvare i contenuti di tali registri.

Operazioni sullo stack di sistema. Ogni volta che l'MC68020 esegue un programma, viene impiegato un solo stack di sistema. Il suo puntatore di stack sarà designato come SP. Nella modalità di utente, SP indica il puntatore dello stack di utente. Invece, il puntatore dello stack di supervisore viene designato come SSP. Non dovrebbe esserci confusione quando si discutono i puntatori di stack o nella programmazione, poiché la modalità operativa della CPU determina l'SP a cui si fa riferimento.

Quando un elemento dev'essere inserito nello stack di sistema da un'istruzione, innanzitutto il puntatore di stack viene decrementato di 2 per un valore di word o di 4 per un operando di 32 bit. Lo stack di sistema si estende sempre verso gli indirizzi più bassi nella memoria, man mano che gli elementi vengono inseriti. Ciò è opposto a quanto avviene per gli stack privati, definiti nel par. 4.2.2, che operano secondo le modalità definite dal programmatore. Dopo il prelievo di un elemento sullo stack, (SP) viene incrementato di 2 o di 4, come richiesto.

Chiamate di subroutine. La Fig. 4.12(a) illustra il flusso di controllo allorché viene chiamata una subroutine. Il programma ha eseguito un'istruzione di salto ad una subroutine — JSR (*Jump to SubRoutine*) o BSR (*Branch to SubRoutine*) — che saranno descritte nel cap. 6. La CPU salva sullo stack di sistema attivo il contenuto di 32 bit del contatore di programma e poi trasferisce il controllo alla subroutine caricando l'indirizzo iniziale nel PC. Quando la subroutine viene completata eseguendo un'istruzione di ritorno RTS (*ReTurn form Subroutine*), la CPU

(a) *Chiamata di subroutine.*

(b) *Eccezione.*

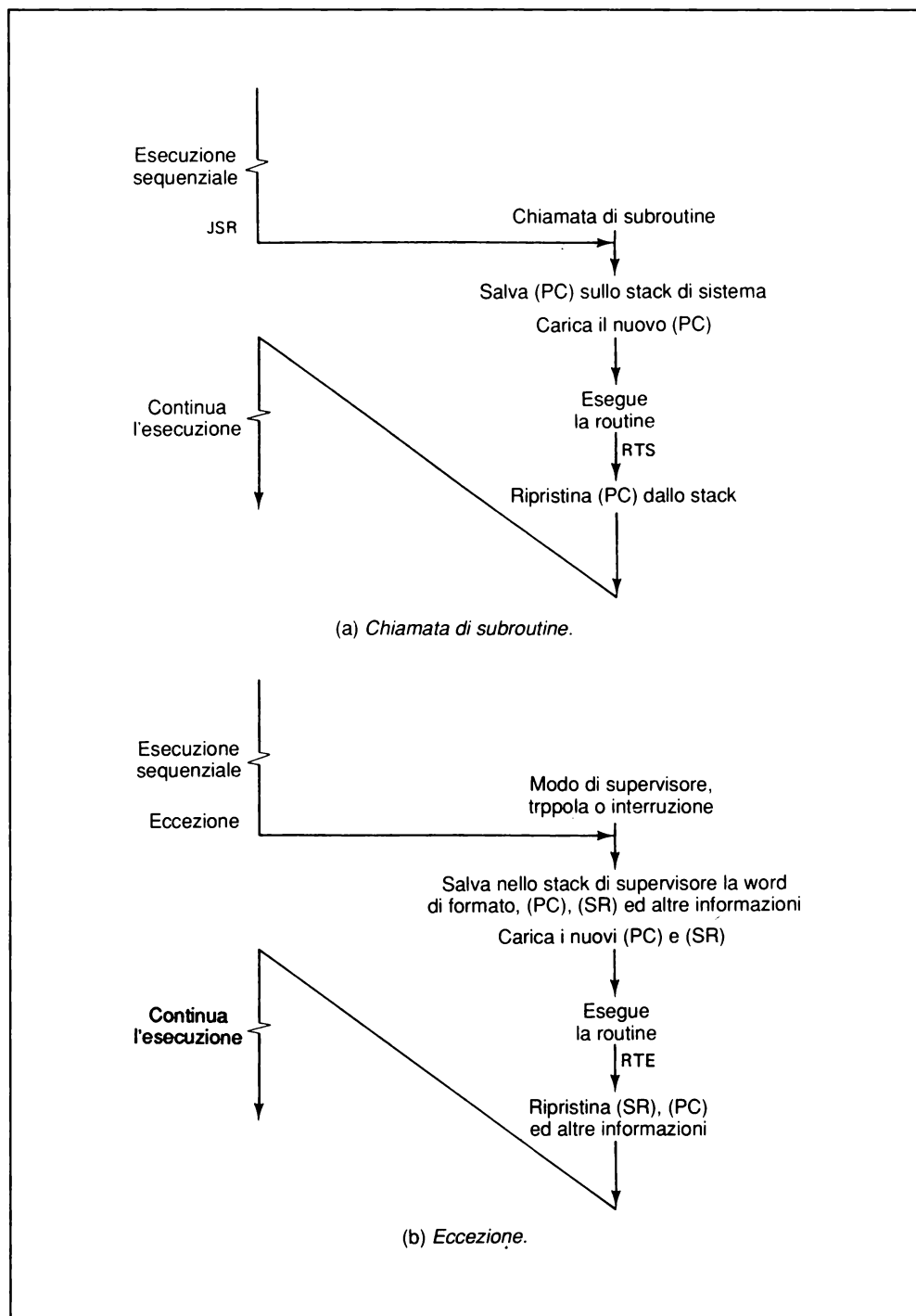



Fig. 4.12 *Operazioni di stack.*

ricarica nel PC il valore che era stato salvato sullo stack. Questo valore è l'indirizzo dell'istruzione successiva a quella che ha causato il salto (JSR o BSR). Se la chiamata alla subroutine era stata effettuata nella modalità di utente, allora viene usato lo stack di utente. Invece, nella modalità di supervisore, l'indirizzo di ritorno viene salvato sullo stack di supervisore. L'esempio 4.6 illustrerà l'impiego di entrambi gli stack per una chiamata di subroutine e per un'interruzione.

Elaborazione delle eccezioni. Quando un'eccezione viene riconosciuta dall'MC68020, la transizione alla routine di gestione dell'eccezione è controllata automaticamente dal modo di supervisore, come mostrato in Fig. 4.12(b). Qualunque eccezione causa la commutazione della CPU al modo di supervisore, indipendentemente dalla modalità in corso nel momento in cui si è presentata l'eccezione. Allorché questa viene riconosciuta, (PC) e (SR), come pure una word (16 bit) di formato, vengono salvati nello stack di supervisore. Per alcune eccezioni è necessario salvare ulteriori informazioni. In qualsiasi caso, saranno disponibili sufficienti informazioni sul programma interrotto da consentirne la ripresa dell'esecuzione allorché la routine di eccezione sarà stata completata. L'ultima istruzione nella routine di eccezione (RTE: *Return form Exception*) causa il ripristino delle informazioni, cosicché il controllo potrà essere restituito al programma originale.

Esempio 4-6

La Fig. 4.13 mostra il contenuto degli stack utilizzati durante l'esecuzione di un programma in modo utente. Se il programma chiama una subroutine, l'indirizzo di ritorno viene inserito nello stack usando (USP), che viene decrementato di 4 per ospitare il (PC) di 32 bit. Se avviene un'interruzione durante l'esecuzione della subroutine, allora vengono salvati usando (SP) sia l'indirizzo di ritorno entro la subroutine interrotta che il contenuto del registro di stato. Il ritorno dalla routine di elaborazione dell'interruzione, seguito dal ritorno finale dalla subroutine, lasciano (USP) e (SSP) così com'erano inizialmente.

4.2.7 Confronto tra le modalità di supervisore e di utente

La differenza tra la modalità di supervisore e quella di utente si basa sul privilegio che riguarda il controllo della CPU stessa ed eventualmente di dispositivi esterni. I programmi eseguiti nel modo di utente sono impossibilitati ad eseguire certe istruzioni e non possono accedere al puntatore di stack di supervisore. I programmi nella modalità di utente possono anche avere il divieto di controllare certi elementi nel sistema del computer o nelle regioni di memoria a cui possono accedere per la lettura o la scrittura di operandi. Queste restrizioni del sistema devono essere imposte da un'apposita circuiteria esterna, non dalla CPU. La CPU indica

(a) *Stack di supervisore.*

SSP → Libera | SSP → Libera | SSP → Stato, Indirizzo di ritorno, Formato | SSP → Libera | SSP → Libera

Indirizzi inferiori di memoria

(b) *Stack di utente.*

USP → Libera | USP → Libera, Indirizzo di ritorno | USP → Libera, Indirizzo di ritorno | USP → Indirizzo di ritorno | USP → Libera

Indirizzi inferiori di memoria

Sequenza di esecuzione:

Programma di utente — (i) Prima della chiamata

Subroutine — (ii) Durante l'esecuzione

Routine d'interruzione — (iii) Durante l'interruzione

Subroutine — (iv) Dopo il ritorno alla subroutine

Programma di utente — (v) Dopo il ritorno dalla subroutine

TEMPO →

Fig. 4.13 *Utilizzazione dello stack durante la chiamata di subroutine e l'elaborazione dell'interruzione.*

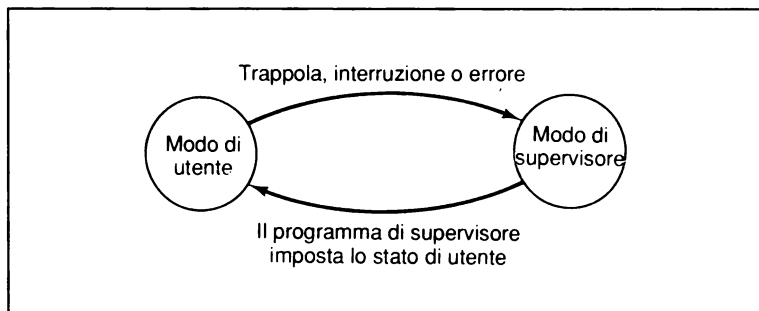
la sua modalità tramite alcune delle sue linee di segnale di controllo, per consentire ai circuiti esterni di determinare l'azione appropriata. Ad esempio, l'unità di gestione della memoria MC68851 può essere utilizzata per limitare l'accesso alla memoria per un programma che opera in modo utente. Qualora tale accesso fosse tentato, l'MC68851 invierebbe alla CPU M68020 un segnale che causerebbe un'eccezione di errore. Il sistema operativo determinerà quindi l'azione appropriata, che potrebbe essere il divieto di proseguire l'esecuzione del programma responsabile del tentativo di accesso vietato. Queste considerazioni di sistema saranno discusse ulteriormente nei capp. 10, 11 e 12.

Distinzioni di privilegio e transizioni. La Tab. 4.3 indica alcune differenze tra la modalità di supervisore e quella di utente. Le due modalità condividono il medesimo insieme di registri generali ed il contatore di programma, ma i puntatori di stack impiegati nei due modi sono distinti. Un programma in modo utente non può eseguire l'insieme di istruzioni completo dell'MC68020. In particolare, le istruzioni per modificare la modalità o per controllare alcune caratteristiche del sistema non sono eseguibili da programmi che operano nella modalità di utente. Queste sono definite *istruzioni privilegiate*. Se venisse effettuato un tentativo di eseguire una istruzione privilegiata da un programma in modo utente, scatterebbe una trappola e il controllo del computer sarebbe restituito al sistema operativo.

I modi di transizione sono accuratamente controllati, come indicato nella Tab. 4.3 e nella Fig. 4.14. Il modo di utente viene attivato soltanto facendo sì che il sistema operativo o il programma in modo supervisore cambi la modalità del processore da quella di supervisore a quella di utente. Ciò si ottiene modificando il registro di stato (con un'istruzione privilegiata), come sarà spiegato nel prossimo sottoparagrafo. Il ritorno nel modo di supervisore ha luogo quando un'eccezione viene riconosciuta dalla CPU.

Tab. 4.3 *Confronto tra modo di supervisore e modo di utente.*

	Supervisore	**Utente**
Registri impiegati	D0-D7, A0-A6, PC, SR	D0-D7, A0-A6, PC, CCR
Puntatore di stack	SSP	USP
Istruzioni	Tutte	Insieme ristretto
Attivata da:	Elaborazione dell'eccezione	Programma di supervisore che commuta la modalità in quella di utente
Registri speciali	Controllo di cache ed altri registri speciali	—

Fig. 4.14
Transizioni tra le modalità.

L'insieme di registri speciali, disponibile soltanto nella modalità di supervisore, include il registro di stato e alcuni altri registri speciali. Per esempio, due di questi registri sono utilizzati per controllare la memoria cache della CPU. Altri due sono i puntatori di stack associati con la modalità di supervisore.

4.2.8 Il registro di stato

La Fig. 4.15 illustra il registro di stato di 16 bit dell'MC68020. Gli 8 bit meno significativi sono denominati *codici di condizione* e possono essere letti o modificati da programmi eseguiti sia nella modalità di supervisore che in quella di utente. Il byte superiore è definito *byte di sistema* e può essere modificato soltanto da un programma nella modalità di supervisore. Ogni bit del byte più significativo è definito nella Tab. 4.4.

Note:
(1) Le condizioni espresse sono vere quando il bit corrispondente è {1}.
(2) Il byte meno significativo del registro di stato è noto come registro dei codici di condizione (CCR).

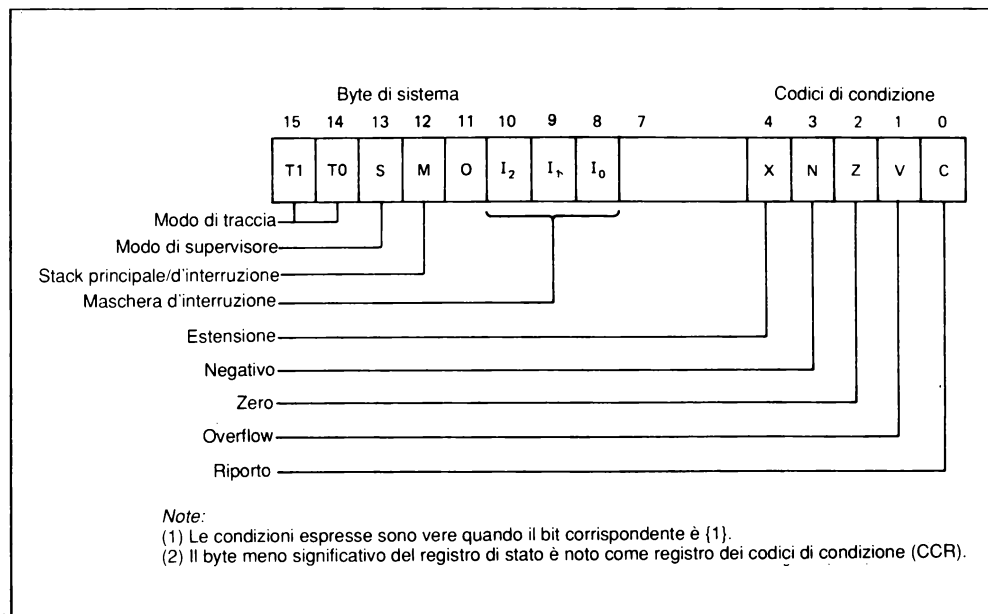



Fig. 4.15 *Il registro di stato dell'MC68020.*

Tab. 4.4 *Interpretazioni dello stato del sistema.*

Nome	Simbolo	Significato
Traccia	T1	Posto a {1} se è impiegata la traccia della singola istruzione.
Traccia	T0	Posto a {1} se è impiegata la traccia del cambiamento del flusso di controllo.
Supervisore	S	Posto a {1} se il programma del processore è eseguito nella modalità di supervisore.
Stack principale	M	Posto a {1} se è attivo lo stack principale; altrimenti, viene usato lo stack d'interruzione per tutte le eccezioni.
Maschera d'interruzione	I0, I1, I2	Livello d'interruzione codificato. Posto a {000} per abilitare tutti i livelli d'interruzione. Posto a {111} per disabilitare tutti i livelli d'interruzione tranne il livello 7.

Nella figura 4.15, ciascun bit è considerato separatamente, tranne i bit di maschera dell'interruzione SR[10:8], che sono raggruppati come un numero binario di 3 bit che indica il livello d'interruzione. Gli altri bit sono considerati singolarmente; in essi, un valore {1} indica che la condizione corrispondente è vera. Quindi la condizione SR[13] = {1} indica che è in esecuzione un programma nel modo di supervisore, mentre SR[13] = {0} indica un programma nel modo di utente. In pratica, il sistema operativo definisce SR[13] = {0} e modifica gli altri bit come necessario allorché il controllo viene passato al programma in modo utente. Se poi si presentasse una condizione di eccezione, allora la CPU operante nel modo di supervisore salverebbe sullo stack di sistema tutte le informazioni pertinenti, inclusi i valori correnti di (PC) e (SR). La condizione di eccezione modifica automaticamente il bit S in {1} e causa le necessarie modifiche degli altri bit del registro di stato. Allorché il controllo viene restituito al programma di utente, i contenuti precedenti del registro di stato e del contatore di programma sono prelevati dallo stack di sistema e ripristinati. Quando viene ripristinato il contenuto precedente del contatore di programma, l'esecuzione del programma di utente riprende dal punto in cui era stata sospesa. Quindi il controllo della modalità della CPU è determinato dal valore assegnato al bit S, il che fornisce un metodo semplice ed efficiente di commutazione tra le modalità.

Come ausilio allo sviluppo del programma, il sistema operativo può generare una "traccia", istruzione per istruzione, quando (SR)[15] è posto a {1}. In alternativa, quando (SR)[14] = {1}, viene causata un'eccezione di traccia allorché il flusso del programma viene cambiato da un'istruzione, come discusso nel cap. 2. Quando viene emessa un'eccezione di traccia, l'esecuzione continua nell'appropriata routine di gestione della traccia, utilizzata come ausilio al debugging.

Il processore o il sistema operativo seleziona uno dei due puntatori di stack di supervisore attivando o azzerando il bit 12. Quando S = {1} e M = {1}, il puntatore di stack principale (*Master Stack Pointer:* MSP) è il puntatore di stack attivo. Invece, se S = {1} e M = {0}, viene selezionato come puntatore di stack di sistema il puntatore di stack d'interruzione (*Interrupt Stack Pointer:* ISP). Lo stack principale sarà descritto nel par. 12.5.

Il sistema d'interruzione (*interrupt*) dell'MC68020 è controllato da 3 bit nel registro di stato. Questi bit agiscono come una maschera d'interruzione per il sistema d'interruzione di priorità a otto livelli dell'MC68020. L'assegnazione di un valore da 0 a 6 a questo gruppo di tre bit disabilita ("maschera") le interruzioni ai livelli indicati e a quelli di priorità inferiore. L'interruzione di livello 7 è nota come *interruzione non mascherabile* perché non può essere disabilitata. Il valore decimale della maschera d'interruzione è da 0 a 7, con valori di priorità in ordine crescente. Se avviene un'interruzione ad un certo livello, allora i bit di maschera a quel livello vengono attivati automaticamente, per impedire la ricezione di ulteriori interruzioni da quel livello o da un livello inferiore.

Esempio 4-7

L'operando di 8 bit {1XXX XXXX} causerebbe la condizione N = {1} se fosse esaminato per un valore negativo. L'impostazione degli altri bit designati con X non ha effetto sul test. A seconda dell'applicazione del programma, l'interruzione potrebbe essere causata da: un numero negativo in complemento a 2, un numero binario senza segno maggiore o uguale a 128, o un numero BCD maggiore o uguale a 80.

Esempio 4-8

Il contenuto 0700_{16} del registro di stato indica la modalità di utente in cui tutti i livelli d'interruzione al di sotto del livello 7 sono mascherati (disabilitati). Tutti i codici di condizione sono {0}. Sarà riconosciuta soltanto un'interruzione di livello 7, poiché risulta non mascherabile. Qualora si presentasse un'interruzione siffatta, il contenuto di (SR) sarebbe posto a 2700_{16} mentre l'interruzione viene elaborata, indicando che l'elaborazione avviene nella modalità di supervisore. Al completamento della routine d'interruzione, il controllo viene riportato al programma in modo utente, con (SR) = 0700_{16}.

ESERCIZI

4.2.1 Si determini lo stato se il registro di stato del sistema contiene i seguenti valori esadecimali:
(a) 0400
(b) 2000
(c) 0004
(d) A000
(e) 7700

4.2.2 S'illustri il contenuto del registro di stato del sistema dopo che un'interruzione di livello 4 è stata accettata. Si supponga che il registro di stato contenesse inizialmente 0 per ciascun bit.

4.2.3 Quali registri devono essere inizializzati prima che il processore possa eseguire un programma? Si consideri dapprima un programma nel modo di supervisore. Quali registri devono essere inizializzati dal sistema operativo prima che il controllo sia passato ad un programma in modo utente?

4.2.4 S'illustri il contenuto dello stack di sistema se un programma viene interrotto da un'interruzione di livello 1 quando (PC) = $101C_{16}$. Se la routine dell'interruzione di livello 1 viene interrotta anch'essa da un'interruzione di livello 2 quando (PC) = $200C_{16}$, si mostrino i cambiamenti che avvengono nello stack di sistema. Si supponga che inizialmente (SR) = 0 e (ISP) = 8000_{16}.

4.3 INTRODUZIONE ALL'INSIEME DI ISTRUZIONI DELL'MC68020

L'insieme di istruzioni dell'MC68020 determina le operazioni disponibili per eseguire trasferimenti di dati ed elaborazioni aritmetiche e per controllare il flusso del programma. Ogni istruzione completa dell'MC68020 consiste delle seguenti parti:

(a) Un codice operativo che determina l'operazione da eseguire;

(b) Una designazione della lunghezza dell'operando o degli operandi;

(c) La specificazione delle locazioni di qualsiasi operando interessato, indicando una modalità d'indirizzamento per ciascuno di essi.

La Fig. 4.16 elenca l'insieme di istruzioni dell'MC68020 in ordine alfabetico. Ogni codice mnemonico rappresenta un codice operativo. Una lettera è usata per indicare una lunghezza di un byte (B), di una word (W), o di una longword (L), per operandi di 8, 16 o 32 bit, rispettivamente. Per esempio, l'istruzione simbolica per sommare due operandi di 16 bit sarebbe:

```
ADD.W     X, Y
```

dove X e Y designano le locazioni degli operandi.

Mnemonico	Descrizione
ABCD	Add Decimal with Extend
ADD	Add
ADDA	Add Address
ADDI	Add Immediate
ADDQ	Add Quick
ADDX	Add with Extend
AND	Logical AND
ANDI	Logical AND Immediate
ASL, ASR	Arithmetic Shift Left and Right
Bcc	Branch Conditionally
BCHG	Test Bit and Change
BCLR	Test Bit and Clear
BFCHG	Test Bit Field and Change
BFCLR	Test Bit Field and Clear
BFEXTS	Signed Bit Field Extract
BFEXTU	Unsigned Bit Field Extract
BFFFO	Bit Field Find First One
BFINS	Bit Field Insert
BFSET	Test Bit Field and Set
BFTST	Test Bit Field
BKPT	Breakpoint
BRA	Branch
BSET	Test Bit and Set
BSR	Branch to Subroutine
BTST	Test Bit
CALLM	Call Module
CAS	Compare and Swap Operands
CAS2	Compare and Swap Dual Operands
CHK	Check Register Against Bound
CHK2	Check Register Against Upper and Lower Bounds
CLR	Clear
CMP	Compare
CMPA	Compare Address
CMPI	Compare Immediate
CMPM	Compare Memory to Memory
CMP2	Compare Register Against Upper and Lower Bounds
DBcc	Test Condition, Decrement and Branch
DIVS, DIVSL	Signed Divide
DIVU, DIVUL	Unsigned Divide
EOR	Logical Exclusive OR
EORI	Logical Exclusive OR Immediate
EXG	Exchange Registers
EXT,EXTB	Sign Extend
ILLEGAL	Take Illegal Instruction Trap
JMP	Jump
JSR	Jump to Subroutine
LEA	Load Effective Address
LINK	Link and Allocate
LSL, LSR	Logical Shift Left and Right
MOVE	Move
MOVEA	Move Address
MOVE CCR	Move Condition Code Register
MOVE SR	Move Status Register
MOVE USP	Move User Stack Pointer
MOVEC	Move Control Register
MOVEM	Move Multiple Registers
MOVEP	Move Peripheral
MOVEQ	Move Quick
MOVES	Move Alternate Address Space

Mnemonico	Descrizione
MULS	Signed Multiply
MULU	Unsigned Multiply
NBCD	Negate Decimal with Extend
NEG	Negate
NEGX	Negate with Extend
NOP	No Operation
NOT	Logical Complement
OR	Logical Inclusive OR
ORI	Logical Inclusive OR Immediate
PACK	Pack BCD
PEA	Push Effective Address
RESET	Reset External Devices
ROL, ROR	Rotate Left and Right
ROXL, ROXR	Rotate with Extend Left and Right
RTD	Return and Deallocate
RTE	Return from Exception
RTM	Return from Module
RTR	Return and Restore Codes
RTS	Return from Subroutine
SBCD	Subtract Decimal with Extend
Scc	Set Conditionally
STOP	Stop
SUB	Subtract
SUBA	Subtract Address
SUBI	Subtract Immediate
SUBQ	Subtract Quick
SUBX	Subtract with Extend
SWAP	Swap Register Words
TAS	Test Operand and Set
TRAP	Trap
TRAPcc	Trap Conditionally
TRAPV	Trap on Overflow
TST	Test Operand
UNLK	Unlink
UNPK	Unpack BCD
ISTRUZIONI DI COPROCESSORE	
cpBcc	Branch Conditionally
cpDBcc	Test Coprocessor Condition, Decrement, and Branch
cpGEN	Coprocessor General Instruction
cpRESTORE	Restore Internal State of Coprocessor
cpSAVE	Save Internal State of Coprocessor
cpScc	Set Conditionally
cpTRAPcc	Trap Conditionally

Fig. 4.16 *L'insieme di istruzioni dell'MC68020. (Per gentile concessione di Motorola, Inc.)*

Le istruzioni per l'MC68020 possono essere classificate in base al tipo o in base al numero degli operandi. Questo numero determina se l'istruzione è classificabile come a un singolo indirizzo o come a doppio indirizzo. La classificazione per tipo raggruppa in categorie le operazioni fondamentali consentite dal processore, come quelle per il trasferimento di dati o quelle per le operazioni aritmetiche.

L'insieme di istruzioni di un processore è talvolta separato in tipi, al fine di confrontarlo con gli insiemi di istruzioni di altri processori. Tale confronto potrebbe servire, per esempio, a preferire un processore con un esteso insieme aritmetico rispetto ad un altro con capacità inferiori, ai fini di una programmazione matematica. La suddivisione in tipi è comoda anche per codificare le istruzioni, poiché questo raggruppamento consente al programmatore di selezionare l'istruzione più adatta per eseguire un'operazione di un certo tipo. Per esempio, l'istruzione di scambio EXG (*EXchanGe*), elencata nella Fig. 4.16, scambia i contenuti di due registri dell'MC68020 ed è più efficiente per questo scopo rispetto ad altre istruzioni di trasferimento di dati che potrebbero conseguire il medesimo risultato.

I tipi fondamentali di istruzioni per l'MC68020 sono quelli di trasferimento di dati, di operazioni logiche e aritmetiche, di controllo del programma e di controllo del processore o del sistema. Queste categorie saranno espanse nel cap. 5, ma per i fini presenti basterà discutere soltanto alcune istruzioni rappresentative dei vari tipi. Per comodità del lettore, le istruzioni sono presentate anche nell'app. D.

4.3.1 L'istruzione CLR

L'istruzione CLR (*CLeaR:* azzera) è considerata un'istruzione aritmetica a un singolo indirizzo; essa ha la seguente forma simbolica:

```
CLR.X       <EAd>
```

dove X è B, L o W, mentre <EAd> rappresenta l'indirizzo effettivo (*Effective Address*) della destinazione. Una sequenza di zeri viene trasferita alla porzione della locazione di destinazione specificata dall'operazione, come mostrato nella Tab. 4.5. Se tale locazione conteneva in origine tutti 1, allora eseguendo l'istruzione CLR.X si otterrà l'azzeramento della porzione designata. L'operazione può essere definita come segue:

$$(EAd)[X] \leftarrow 0[X]$$

che significa: "Il contenuto della locazione EAd di lunghezza X viene sostituito da zeri".

Tab. 4.5 *Operazione dell'istruzione CLR.*

ISTRUZIONE		CONTENUTO DELLA DESTINAZIONE DOPO CHE L'ISTRUZIONE È STATA ESEGUITA
CLR.B	<EAd>	FFFF FF00
CLR.W	<EAd>	FFFF 0000
CLR.L	<EAd>	0000 0000

Nota: La destinazione contiene FFFF FFFF prima dell'esecuzione di ogni istruzione.

La descrizione dell'istruzione CLR dell'MC68020 è mostrata nella Fig. 4.17. Questo sommario, tratto dallo *User's Manual* della Motorola, presenta le caratteristiche dell'istruzione in vari modi. La voce *Operation* (operazione) indica che la locazione di destinazione viene rimpiazzata con zeri. L'assemblatore riconosce il codice mnemonico CLR e converte in linguaggio-macchina tale codice e l'indirizzo effettivo della locazione di destinazione. Le destinazioni valide, in termini delle possibili modalità d'indirizzamento (*Addr. Mode*), sono elencate nella tabella che accompagna la descrizione dell'istruzione. Queste modalità saranno discusse nel par. 4.4. Sono fornite anche altre importanti caratteristiche, come l'effetto del formato di linguaggio-macchina sui codici di condizione.

Esiste un certo numero di modi per specificare la locazione che dovrà essere azzerata dall'istruzione CLR. Il metodo viene scelto tra le undici modalità d'indirizzamento valide mostrate nella tabella in basso nella figura. La lunghezza dell'operando nella locazione di destinazione è indicata come *size* (dimensione) ed è specificata come byte (8 bit), word (16 bit) o longword (32 bit), con le corrispondenti notazioni simboliche B, W o L, rispettivamente. Per esempio, l'assemblatore riconosce

```
CLR.B     D1
```

come l'istruzione per azzerare 8 bit del registro D1. Per descrivere precisamente questa operazione, la notazione adottata in questo libro è la seguente:

$$(D1)[7:0] \leftarrow 0$$

che indica l'azzeramento dei bit da 0 a 7 del registro D1. Il simbolo di sostituzione ($\leftarrow$) significa che l'operando a sinistra della freccia viene sostituito dal valore a destra. Dopo che l'istruzione sarà stata eseguita, il contenuto di una parte della locazione di destinazione sarà azzerato; in questo esempio, ciò viene indicato dall'espressione simbolica:

$$(D1)[7:0] = 0$$

Operation: 0 → Destination

Assembler Syntax: CLR <ea>

Attributes: Size = (Byte, Word, Long)

Description: The destination is cleared to all zero. The size of the operation may be specified to be byte, word, or long.

Condition Codes:

X	N	Z	V	C
—	0	1	0	0

N Always cleared.
Z Always set.
V Always cleared.
C Always cleared.
X Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	1	0	Size		Effective Address Mode			Effective Address Register		

Instruction Fields:

Size field — Specifies the size of the operation.
00—byte operation.
01—word operation.
10—long operation.

Effective Address field — Specifies the destination location. Only data alterable addressing modes are allowed as shown:

Modo Indir	Modo	Registro
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
–(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Modo Indir	Modo	Registro
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

Fig. 4.17 *Descrizione dell'istruzione CLR. (Per gentile concessione di Motorola, Inc.)*

Esempio 4-9

Di seguito sono forniti vari esempi dell'istruzione CLR. Gli indirizzi per le locazioni di destinazione nella memoria sono indicati come valori decimali. Ciò è conforme con la notazione del linguaggio assembler che sarà discussa nel cap. 5. Una locazione di word (16 bit) nella memoria consiste di due byte consecutivi.

FORMA SIMBOLICA DELL'ISTRUZIONE	DOPO L'ESECUZIONE
CLR.B D1	(D1)[7:0] = 0
CLR.W D1	(D1)[15:0] = 0
CLR.B 1000	(1000) = 0
CLR.W 1000	(1000) = 0 (1001) = 0

4.3.2 L'istruzione MOVE

L'istruzione fondamentale di trasferimento dei dati per l'MC68020 è l'istruzione MOVE, che è un'istruzione a doppio indirizzo scritta in forma simbolica come:

```
MOVE.X      <EAs>,<EAd>
```

dove X vale B, W o L e specifica la lunghezza o dimensione dell'operando. Una copia dell'operando di sorgente (provenienza), di lunghezza X, contenuto nella locazione < EAs> , viene trasferita alla locazione di destinazione < EAd> , lasciando invariato il contenuto della locazione di sorgente. Sia l'operando di destinazione che quello di sorgente sono trattati come se fossero di lunghezza X. L'istruzione MOVE copia l'operando di sorgente nei bit [7:0], [15:0] o [31:0] per il trasferimento di un byte, di una word o di una longword, rispettivamente. Gli indirizzi effettivi, <EAs> e <EAd> , vengono calcolati dal processore in conformità con la specificazione della modalità d'indirizzamento. Essi possono denotare registri del processore o locazioni di memoria. La Fig. 4.18 illustra l'operazione dell'istruzione MOVE per le tre lunghezze di operando. in ciascun caso, le locazioni di destinazione e di sorgente contengono valori di 32 bit, ma soltanto la porzione specificata dell'operando viene copiata dalla locazione di sorgente a quella di destinazione.

Longword 32 bit
Word 16 bit
Byte 8 bit
31 16 15 8 7 0

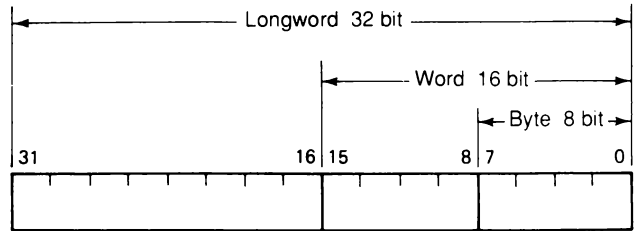
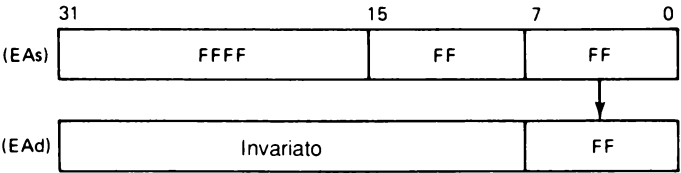
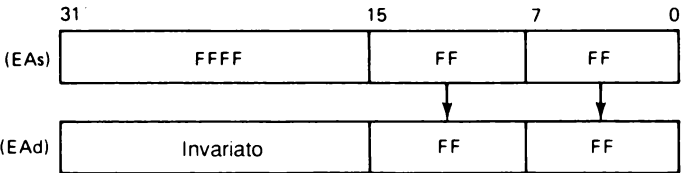
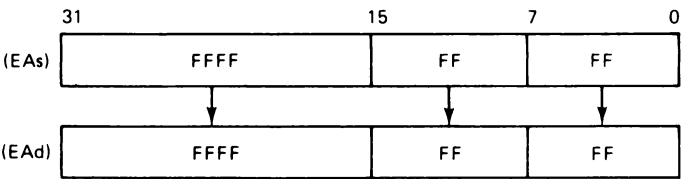
(a) *Operandi di lunghezza di byte. word o longword.*

31 15 7 0
(EAs) FFFF FF FF
(EAd) Invariato FF

(b) MOVE.B

31 15 7 0
(EAs) FFFF FF FF
(EAd) Invariato FF FF

(c) MOVE.W

31 15 7 0
(EAs) FFFF FF FF
(EAd) FFFF FF FF

(d) MOVE.L

Nota: L'operando di sorgente è FFFF $FFFF_{16}$.

Fig. 4.18 *Operazione dell'istruzione MOVE.*

Esempio 4-10

Il seguente sommario mostra alcuni di esempi dell'istruzione MOVE. In ogni caso, la locazione di sorgente — il registro di dati D2 — contiene il valore esadecimale 0FFF 0105 prima dell'esecuzione di ciascuna istruzione. Il registro di destinazione D1 contiene 1000 0000 in 32 bit, prima dell'esecuzione.

ISTRUZIONE		DESTINAZIONE DOPO L'ESECUZIONE	
MOVE.B	D2,D1	1000	0005
MOVE.W	D2,D1	1000	0105
MOVE.L	D2,D1	0FFF	0105

4.3.3 L'istruzione ADD

Un'importante istruzione aritmetica è ADD. Essa ha la forma seguente:

ADD.X <EAs>,<EAd>

ed esegue l'addizione binaria dell'operando di sorgente e dell'operando di destinazione di lunghezza X. In una siffatta istruzione a doppio indirizzo che calcola un risultato, quest'ultimo viene memorizzato nella locazione di destinazione conformemente alla sostituzione

$$(EAd)[X] \leftarrow (EAs)[X] + (EAd)[X]$$

allorché l'istruzione viene eseguita. L'operando di sorgente non viene modificato da un'istruzione di questo tipo. Le istruzioni ADD, CLR e MOVE discusse in precedenza sono tipiche dell'MC68020 per le operazioni aritmetiche o i trasferimenti di dati. In tali istruzioni, devono essere specificate la locazione di un operando e la sua lunghezza, come pure l'operazione da eseguire.

Esempio 4-11

Si supponga che il registro di dati D2 contenga il valore esadecimale 0FFF 0105 e che D1 contenga 1000 0001 prima dell'esecuzione di ciascuna istruzione. I risultati memorizzati in D1 sono mostrati nella seguente tabella per l'addizione di operandi di lunghezza specificata.

ISTRUZIONE	DESTINAZIONE DOPO L'ESECUZIONE	
ADD.B D2,D1	1000	0006
ADD.W D2,D1	1000	0106
ADD.L D2,D1	1FFF	0106

4.3.4 Altri tipi di istruzioni

Le istruzioni di controllo del programma possono modificare il flusso del controllo in un programma, modificando il valore del contatore di programma e quindi causando l'esecuzione di una nuova sequenza di istruzioni. Per esempio, l'istruzione di salto "Jump"

```
JMP          <EA>
```

causa il trasferimento del controllo del programma all'istruzione contenuta nella locazione designata dall'indirizzo effettivo < EA> . L'istruzione di salto "Branch"

```
BRA          <spost>
```

aggiunge un valore di spostamento (*displacement*) al contenuto del contatore di programma nell'istante in cui l'istruzione viene eseguita. Ciò causa il trasferimento del controllo del programma entro l'intervallo ammesso dal valore <spost>, che può essere un intero positivo o negativo. Sia l'istruzione BRA (*Branch Always:* salta comunque) che JMP (*Jump*: salta) causano un trasferimento incondizionato del controllo. Altre istruzioni di salto possono causare effettivamente un salto oppure no, a seconda delle condizioni prodotte da un'operazione aritmetica. Per esempio:

```
BGT          <spost>
```

produce un salto se il risultato era maggiore di zero. La condizione è indicata dai valori dei bit dei codici di condizione nel registro di stato. Queste istruzioni di controllo del programma saranno discusse in dettaglio nel cap. 6.

Le istruzioni che controllano l'attività del processore o del sistema sono generalmente riservate a programmi operanti nel modo di supervisore. Per esempio, l'istruzione STOP fa sì che il processore sospenda il prelievo e l'esecuzione delle istruzioni. Tali istruzioni saranno discusse più dettagliatamente nel cap. 10.

ESERCIZI

4.3.1 Si descriva il contenuto di ciascuna locazione di byte interessata dall'istruzione CLR.L 1000.

4.3.2 Si supponga che, prima dell'esecuzione di ognuna delle istruzioni specificate di seguito, D1 e D2 contengano i seguenti valori esadecimali e che la locazione 1000 sia una longword.

(D1) = 0601
(D2) = 0805
(1000) = 1913

Si determinino i risultati dell'esecuzione di ciascuna delle seguenti istruzioni:

```
(a) MOVE.B   D1,D2
(b) MOVE.B   D1, 1001
(c) CLR.W    1000
(d) ADD.B    D1,D2
(e) ADD.W    1000,D1
```

4.4 MODALITA' D'INDIRIZZAMENTO PER L'MC68020

Le *modalità d'indirizzamento* di una CPU determinano i modi in cui un processore può far riferimento ad un operando contenuto in uno dei suoi registri o nella memoria. Per ciascun operando, la modalità d'indirizzamento specifica il modo in cui il processore deve individuare o calcolare l'indirizzo effettivo dell'operando. Tale *indirizzo effettivo* viene determinato allorché viene eseguita l'istruzione che fa riferimento all'operando.[6]

Le ampie categorie d'indirizzamento per l'MC68020 includono l'indirizzamento diretto, l'indirizzamento indiretto, e l'indirizzamento relativo al contatore di programma. È disponibile anche un modo immediato speciale. La Tab. 4.6 definisce queste modalità fondamentali per l'MC68020. Nella tabella sono elencate la categoria del modo d'indirizzamento e l'indirizzo effettivo che risulta dall'esecuzione dell'istruzione. La designazione simbolica è il riferimento alla modalità d'indirizzamento assegnata, così com'è riconosciuta da un assemblatore della Motorola. Un indirizzo assoluto è considerato come un valore decimale, a meno che non sia preceduto dal simbolo "$" per indicare che si tratta di un valore esadecimale. Il simbolo "*" in un'istruzione simbolica fa riferimento al valore corrente del contatore di programma, mentre il simbolo "#" anteposto ad un numero indica l'indirizzamento immediato.

[6] Quando la CPU indirizza direttamente la memoria, un indirizzo effettivo è l'indirizzo fisico dell'hardware. In molti sistemi avanzati, s'impiegano circuiti speciali che costituiscono la cosiddetta circuiteria di rappresentazione o "mappatura" della memoria (*memory mapping circuitry*). Questa circuiteria calcola quindi l'indirizzo fisico che corrisponde all'indirizzo della CPU. Gli indirizzi fisici in questione dipendono interamente dal progetto del sistema, ma sono indipendenti dai riferimenti di programmazione ad operandi nella memoria. La capacità di rappresentazione della memoria della Memory Management Unit MC68851 della Motorola sarà descritta nel par. 12.4.

Tab. 4.6 *Modalità d'indirizzamento fondamentali.*

Tipo	Indirizzo effettivo	Designazione simbolica
DIRETTO		
Registro	EA = Rn	D0, D1, ..., D7; A0, A1, ..., A7
Assoluto	EA = <Indirizzo>	<indirizzo decimale> $<indirizzo esadecimale>
INDIRETTO		
Registro d'indirizzo	EA = (An)	(A0), (A1), ..., (A7)
Predecremento	An = An – k, EA = (An)	–(An)
Postincremento	EA = (An) An = An + k	(An)+
Memoria	EA = ([An])	([An])
RELATIVO CON SPOSTAMENTO	EA = (PC) + <spost>	* + <spost> o (<spost>, PC)
IMMEDIATO	Nessuno	#<dato>

Note:
1. Rn denota un qualsiasi registro Dn o An.
2. Per i modi di predecremento e postincremento, k vale 1, 2 o 4 per operazioni su byte, word o longword, rispettivamente.
3. Le parentesi angolari <> implicano che dev'essere specificato il valore indicato.
4. La notazione [An] indica un indirizzamento indiretto di memoria.

Le istruzioni dell'MC68020 possono specificare uno o due operandi nella maniera descritta nel par. 4.3 L'istruzione CLR, per esempio, può specificare la designazione mediante una qualsiasi della modalità indicate nella Tab. 4.6, tranne quella immediata. Le istruzioni MOVE e ADD richiedono due operandi e devono essere specificate le modalità d'indirizzamento sia per la sorgente che per la destinazione. Un certo numero di esempi in questo paragrafo illustreranno il modo in cui sono specificate simbolicamente le modalità d'indirizzamento fondamentali. La discussione presentata in questo paragrafo è limitata a quelle modalità mostrate nella Tab. 4.6, che rappresentano soltanto 8 delle 18 possibili modalità d'indirizzamento per l'MC68020. Uno studio più dettagliato dei modi d'indirizzamento dell'MC68020 sarà fornito nel cap. 5, dopo aver introdotto la programmazione in linguaggio assembler.

4.4.1 Indirizzamento diretto

Le modalità d'indirizzamento *diretto* dell'MC68020 comprendono l'indirizzamento di registro e l'indirizzamento assoluto. In entrambi i casi, è specificata direttamente la locazione o l'indirizzo di un operando come parte dell'istruzione, per cui non è necessario alcun calcolo d'indirizzo effettivo da parte della CPU. Per le modalità di registro, l'operando è uno dei registri d'indirizzo o di dati. Nel modo assoluto, l'operando è situato nella memoria, in una locazione designata da un intero positivo che ne rappresenta l'indirizzo.

Il formato fondamentale per l'istruzione CLR che impiega l'indirizzamento di registro è il seguente:

```
CLR.<X>     <Dn>
```

in cui l'operando di lunghezza X è azzerato nel registro Dn, che viene scritto esplicitamente come uno dei registri D0, D1, ..., D7. Quindi l'istruzione

```
CLR.W       D2
```

azzera i 16 bit meno significativi del registro D2. L'istruzione MOVE richiede due operandi ed ha la forma:

```
MOVE.<X> <Dm>,<Dn>
```

come in:

```
MOVE.W      D1,D2
```

che copia (D1)[15:0] in (D2)[15:0].

Un indirizzo assoluto può essere specificato come un intero decimale o esadecimale in un'istruzione. Per esempio, l'istruzione:

```
MOVE.W      10000,D1
```

trasferisce 16 bit dalla locazione di word 10000 in (D1)[15:0]. In base alle convenzioni adottate dagli assemblatori della Motorola, la forma simbolica per la medesima locazione in esadecimale sarebbe:

```
MOVE.W      $2710,D1
```

poiché il valore 2710_{16} corrisponde a 10000 e il "$" indica "esadecimale".

Esempio 4-12

Poiché i modi d'indirizzamento sono quelli più semplici, essi sono stati impiegati nel paragrafo precedente per presentare importanti istruzioni del processore. Per esempio, l'istruzione

```
CLR.W      1000
```

specifica l'indirizzo assoluto 1000 come destinazione. Tale indirizzo viene registrato con l'istruzione nella memoria. L'istruzione

```
MOVE.W     1000, D1
```

impiega l'indirizzamento assoluto per la locazione di sorgente e l'indirizzamento diretto di registro per la destinazione. Un'istruzione quale:

```
MOVE.W     A1,D1
```

trasferisce l'indirizzo di 16 bit in A1 al registro di dati designato come D1.

4.4.2 Indirizzamento indiretto

Nell'MC68020, l'indirizzamento *indiretto* di registro implica l'uso del contenuto di un registro d'indirizzo come indirizzo di un operando nella memoria. Tale contenuto è impiegato come un puntatore per far riferimento alla locazione. Ad esempio, se l'istruzione

```
MOVE.W     (A1),D1
```

viene eseguita quando (A1) = 1000, allora il valore di 16 bit nella locazione 1000 di word nella memoria sarebbe copiato in (D1)[15:0]. Per modificare l'indirizzo cui si fa riferimento nella memoria, il registro d'indirizzo dev'essere modificato da qualsiasi istruzione che operi sul contenuto dei registri d'indirizzo. Questa capacità di modificare il puntatore in maniera così flessibile consente ad un programmatore d'indirizzare valori in strutture di dati "sofisticate" nella memoria. Un semplice esempio è la struttura di stack discussa nel par. 4.2.3.

Lo stack. In effetti, la struttura di stack è talmente comune nei moderni programmi che l'MC68020 ha due modi d'indirizzamento indiretto che sono usati principalmente con gli stack. Questi stack privati possono essere creati ed utilizzati impiegando i modi d'indirizzamento indiretti di registro d'indirizzo con postincremento o predecremento. Ad esempio, per aggiungere dati ad uno stack che cresce da indirizzi alti verso indirizzi bassi nella memoria, l'istruzione

```
MOVE.W     D1,-(A1)
```

trasferisce una word da D1 allo stack dopo che il puntatore di stack (A1) è stato decrementato di due (byte) per puntare alla successiva locazione libera nella memoria. Un elemento di dati potrebbe essere prelevato con l'istruzione

MOVE.W (A1)+,D2

che preleva la word dallo stack indirizzato da A1 e la copia in (D2)[15:0]. Dopo il trasferimento, A1 viene incrementato di 2. L'operazione d'inserimento, per lo stack che cresce vero il basso, impiega il modo d'indirizzamento con predecremento, usando A1 come puntatore di stack. Il prelievo richiede il modo con postincremento per la modalità d'indirizzamento della sorgente. Qualsiasi registro d'indirizzo generale dell'MC68020 può essere usato come puntatore di stack privato. La locazione della sorgente in un inserimento o la locazione di destinazione in un prelievo può essere una locazione di memoria o un registro qualsiasi dell'MC68020. Se A7 è designato come un puntatore di stack, allora l'operazione riguarderebbe lo stack di sistema.

Indirizzamento indiretto della memoria. L'MC68020 consente di utilizzare un valore nella memoria come un indirizzo indiretto per individuare un operando. Un registro d'indirizzo viene usato per puntare alla locazione in cui è contenuto l'indirizzo dell'operando. Quindi, come esempio, si supponga che l'istruzione

MOVE.W ([A1]),D1

sia eseguita con (A1) = 1000 e che il valore conenuto nella locazione di longword 1000 sia 500. Allora il valore di 16 bit nella locazione 500 viene trasferito a D1[15:0]. Il risultato è:

(D1)[15:0] = (500) = ((1000)) = (((A1)))

Per un confronto, l'istruzione

MOVE.W (A1),D1

impiega il modo d'indirizzamento indiretto di registro e rende (D1)[15:0] = 500 = ((A1)) dopo l'esecuzione.

Nel modo d'indirizzamento indiretto della memoria, devono essere forniti due indirizzi. Primo, l'indirizzo l'indirizzo della locazione che contiene l'indirizzo dell'operando dev'essere contenuto in un registro d'indirizzo, Secondo, l'indirizzo effettivo dell'operando dev'essere nella locazione di memoria puntata dal registro d'indirizzo suddetto.

4.4.3 Indirizzamento relativo

Un indirizzo *relativo* al contatore di programma è un indirizzo che la CPU calcola aggiungendo uno spostamento al valore del contatore di programma. L'indirizzo effettivo calcolato è allora:

EA = (PC) + <spost>

in cui il valore <spost> dello spostamento è specificato nell'istruzione. Lo spostamento è un intero positivo o negativo, cosicché la locazione riferita può essere maggiore o inferiore nella memoria relativamente all'istruzione che utilizza questa modalità d'indirizzamento.

Un esempio d'indirizzamento relativo è indicato dall'istruzione:

```
BRA     * + 10
```

la cui esecuzione causerebbe un salto di 10 locazioni di byte più avanti rispetto a quella indicata dal contatore di programma.[7] Nel caso dell'istruzione BRA, il valore nel contatore di programma viene modificato al nuovo indirizzo, per puntare all'istruzione successiva, sei locazioni di word più in alto nella memoria dalla locazione dell'istruzione BRA. L'indirizzamento relativo può essere usato anche per indirizzare valori di dati nella memoria.

Poiché il contenuto del contatore di programma agisce come un puntatore all'istruzione in corso di esecuzione, il valore dello spostamento indica la distanza tra l'operando riferito nella modalità relativa e l'istruzione stessa. Se il programma viene trasferito nella memoria, i riferimenti relativi nel programma sono ancora corretti. Quando i riferimenti alla memoria usati da un programma sono relativi, si dice che il programma è *indipendente dalla posizione*. I programmi di questo tipo saranno discussi nel cap. 9. I programmi con riferimenti assoluti alle locazioni di memoria non possono essere trasferiti, a meno che non vengano modificati gli indirizzi assoluti per indicare le nuove locazioni.

L'MC68020 consente anche l'indirizzamento relativo al contatore di programma con indicizzazione. In questa modalità, l'indirizzo effettivo viene calcolato come contenuto del PC, più un valore di spostamento, più il contenuto di un registro indice. Sono possibili anche riferimenti indiretti alla memoria che impiegano il PC. Tali varianti dell'indirizzamento relativo saranno discusse nel cap. 5.

4.4.4 Indirizzamento immediato

La modalità d'indirizzamento *immediato* è usata per specificare una costante lunga 8, 16 o 32 bit. Tale costante è inclusa nell'istruzione nella memoria. Per esempio, l'istruzione:

```
ADD.W     #5,D1
```

aggiunge 5 al valore in (D1)[15:0].

[7] Quando viene intrapreso il salto, il valore del PC è l'indirizzo dell'istruzione BRA + 2. Questa istruzione BRA richiede due locazioni di word nella memoria; pertanto, essa causa un salto ad un'istruzione situata sei locazioni di word più in alto nella memoria, rispetto alla prima word dell'istruzione BRA stessa. Una forma alternativa è BRA (10,PC).

L'istruzione

```
MOVE.B    #'A',(A1)
```

trasferisce il valore ASCII 'A' nel byte indirizzato da A1 nella memoria. L'assemblatore riconosce come immediati i modi d'indirizzamento della sorgente in questi due esempi. Naturalmente, il modo immediato non è mai ammesso come un modo di destinazione, poiché la locazione di destinazione dev'essere modificabile (per consentire la scrittura).

Esempio 4-13

L'istruzione simbolica

```
MOVE.L    #'1234',D1
```

causa la sostituzione del contenuto di D1 con l'equivalente ASCII di 1234 o col valore esadecimale 3132 3334. Similmente, l'istruzione

```
MOVE.W    #$F0,D1
```

ha l'effetto

$(D1)[15:0] \leftarrow F0_{16}$

Un'istruzione avente un'operando immediato come destinazione, ad esempio:

```
MOVE.B    1000,#1000
```

non sarebbe ammessa e non potrebbe essere assemblata.

ESERCIZI

4.4.1 Usando valori esadecimali per tutte le risposte, si determinino le operazioni svolte e le locazioni interessate da ciascuna delle seguenti istruzioni:

```
(a) MOVE.W    1000,2000
(b) MOVE.W    $1000,D1
(c) MOVE.B    1000,D1
(d) CLR.L     $FFFFFC
```

4.4.2 Si confrontino le operazioni delle seguenti istruzioni per il caso in cui (A1) = 1000 e (100) = $FFE0_{16}$ prima di ogni esecuzione:

```
(a) MOVE.W    A1,D1
(b) MOVE.W    (A1),D1
(c) MOVE.W    1000,D1
(d) MOVE.W    #1000,D1
```

4.4.3 Si determini il contenuto della destinazione, esprimendolo in esadecimale, dopo l'esecuzione di ciascuna delle seguenti istruzioni:

```
(a) MOVE.W   #'AB',D1
(b) MOVE.W   #$C1,D1
(c) MOVE.W   #1000,D1
```

4.4.4 Impiegando soltanto le istruzioni e i metodi discussi finora, si scrivano le istruzioni simboliche per memorizzare la word meno significativa di D1 nelle locazioni di memoria 1001 e 1002, cioè, dopo l'esecuzione, (1001) = (D1)[15:8] e (1002) = (D1)[7:0].

(a) Si supponga che gli operandi di lunghezza di word debbano iniziare in locazioni pari della memoria e che ognuno di essi occupi due byte.

(b) Nell'MC68020, gli operandi di lunghezza di word non devono essere memorizzati in locazioni d'indirizzo pari. Si scriva la singola istruzione per eseguire il trasferimento per l'MC68020.

4.4.5 Si supponga che (A1) = $11000 e che ($11000) = $30000 e ($30000) = $00001000 prima dell'esecuzione di ciascuna delle seguenti istruzioni. I valori sono tutti di longword. Si determini il valore presente in D1 dopo l'esecuzione delle istruzioni (a) e (b) definite di seguito:

```
(a) MOVE.W    [(A1)],D1
(b) MOVEA.L   [(A1)],A2
    MOVE.W    (A2),D1
```

Qui MOVEA è il codice mnemonico usato quando la destinazione è un registro d'indirizzo. Si assuma che ($1000) = $00F05BFE.

4.5 LINGUAGGIO-MACCHINA PER L'MC68020

Le istruzioni in linguaggio-macchina per l'MC68020 consistono di un certo numero di word (da 1 a 16) nella memoria. La prima word è quella dell'operazione; essa contiene il codice operativo (cod.op.), come pure la dimensione o lunghezza e i modi d'indirizzamento per qualsiasi operando, se necessari. Per la maggior parte delle istruzioni, il cod.op. è contenuto nei bit da 12 a 15 della prima word. Varie combinazioni di questi quattro bit forniscono 16 distinti codici operativi.

Il significato di ciascuno di essi è definito nella Fig. 4.19. I restanti 12 bit nella word dell'operazione servono a definire ulteriormente l'operazione da eseguire. Ulteriori word di estensioni per le istruzioni in linguaggio-macchina possono contenere dati immediati o indrizzi assoluti per gli operandi di sorgente o di destinazione. Un indirizzo assoluto breve (16 bit) richiede una word extra, mentre un indirizzo lungo (32 bit) ne richiede due. Il formato dell'istruzione in linguaggio-macchina è mostrato nella Fig. 4.20. Le word di estensione seguono il codice operativo ad indirizzi di memoria superiori.

In questo paragrafo sono discussi i formati delle istruzioni a singolo indirizzo e a doppio indirizzo. Come già spiegato in precedenti paragrafi di questo capitolo, saranno impiegate come esempi specifici le istruzioni CLR, ADD e MOVE. Nell'app. D sono descritte tutte le istruzioni dell'MC68020.

Fig. 4.19
Codici operativi. (Per gentile concessione di Motorola, Inc.)

Bit 15-12	Operazione
0000	Manipolazione di bit/MOVEP/Immediato
0001	Trasferimento di byte
0010	Trasferimento lungo
0011	Trasferimento di word
0100	Miscellanea
0101	ADDQ/SUBQ/Scc/DBcc/TRAPcc
0110	Bcc/BSR/BRA
0111	MOVEQ
1000	OR/DIV/SBCD
1001	SUB/SUBX
1010	(Non assegnata, Riservata)
1011	CMP/EOR
1100	AND/MUL/ABCD/EXG
1101	ADD/ADDX
1110	Scorrimento/Rotazione/Campo di bit
1111	Interfaccia di coprocessore

15 ... 0
Operazione di word (Una sola word; specifica l'operazione e le modalità)
Specificatori di operandi speciali (Se esistono, una o due word)
Operando immediato o estensione d'indirizzo effettivo di sorgente (Se esiste, da una a cinque word)
Estensione d'indirizzo effettivo della destinazione (Se esiste, da una a cinque word)

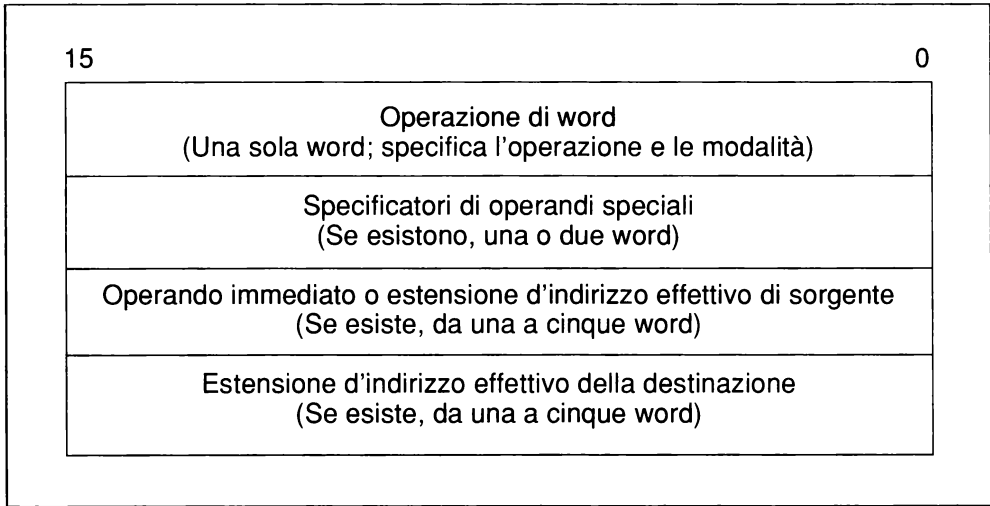



Fig. 4.20 *Formati di istruzione. (Per gentile concessione di Motorola, Inc.)*

4.5.1 Istruzioni a singolo indirizzo

La word di operazione per un'istruzione a un singolo indirizzo è mostrata nella Fig. 4.21(a). I bit [15:6] definiscono l'operazione, mentre i bit [5:0] designano la modalità d'indirizzamento. Il campo d'indirizzo effettivo è suddiviso a sua volta nei

15	14	13	12	11	10	9	8	7	6	5 4 3	2 1 0
										Indirizzo effettivo	
X	X	X	X	X	X	X	X	X	X	Modo	Registro

(a) *Formato generale.*

15	14	13	12	11	10	9	8	7 6	5 4 3	2 1 0
									Indirizzo effettivo	
0	1	0	0	0	0	1	0	Size	**Modo**	**Registro**

Campi di istruzione: Dimensione del campo -
Specifica la dimensione dell'operazione:

00 - operazione di byte
01 - operazione di word
10 - operazione di longword

(b) *L'istruzione CLR.*

0100 0010	00	000	001
Cod.op.	Dimensione	Modo	Registro
		Indirizzo effettivo	

CLR.B D1

(c) *Esempio di istruzione CLR.*

Fig. 4.21 *Istruzioni a un singolo indirizzo. (Per gentile concessione di Motorola, Inc.)*

sottocampi di modo e registro, di 3 bit ciascuno. Per una modalità d'indirizzamento che impiega un registro, il numero del registro (0-7) è specificato nel sottocampo di registro. In questo caso, il sottocampo del modo specifica se s'impiega l'indirizzamento diretto o quello indiretto, per cui valgono le varianti mostrate nella Fig. 4.22. Le modalità d'indirizzamento assoluto, relativo o immediato hanno una codifica fissa per l'intero campo di 6 bit.

Come esempio, l'istruzione

```
CLR.B        D1
```

specifica la destinazione D1 per indirizzamento diretto di registro. Il formato di linguaggio-macchina è mostrato nella Fig. 4.21(b). Per i registri di dati, il modo nel campo d'indirizzo effettivo è {000} ed il numero di registro è {001}. La configurazione dei bit [15:8] specifica l'istruzione CLR. In questo esempio, la dimensione dell'operando è di un byte ed è indicata da 00 nei bit [7:6]. Poiché viene usato soltanto l'indirizzamento di registro, l'istruzione richiede soltanto una word della memoria.

Modo d'indirizzamento	Modo	Registro
Diretto di registro dati	000	num. reg.
Diretto di registro indirizzo	001	num. reg.
Indiretto di registro indirizzo	010	num. reg.
Indiretto di registro indirizzo con postincremento	011	num. reg.
Indiretto di registro indirizzo con predecremento	100	num. reg.
Indiretto di registro indirizzo con spostamento	101	num. reg.
Indiretto di registro indirizzo con indice (spostamento di 8 bit)	110	num. reg.
Indiretto di registro indirizzo con indice (spostamento di base)	110	num. reg.
Indiretto di memoria postindicizzato	110	num. reg.
Indiretto di memoria preindicizzato	110	num. reg.
Assoluto corto	111	000
Assoluto lungo	111	001
Indiretto di contatore di programma con spostamento	111	010
Indiretto di contatore di programma con indice (spostamento di 8 bit)	111	011
Indiretto di contatore di programma con indice (spostamento di base)	111	011
Indiretto di memoria di PC postindicizzato	111	011
Indiretto di memoria di PC preindicizzato	111	011
Immediato	111	100

Fig. 4.22 *Codifica dell'indirizzo effettivo. (Per gentile concessione di Motorola, Inc.)*

Un certo numero di altre istruzioni a un singolo indirizzo, quali NEG (negazione), NOT (complemento a 1) e NBCD (negazione di decimale), hanno il medesimo formato generale dell'istruzione CLR. Altre istruzioni con singoli operandi o quelle senza operandi possono avere un formato di linguaggio-macchina considerevolmente diverso da quello mostrato per l'istruzione CLR. Le istruzioni per il controllo del processore possono essere prive di specificazioni d'indirizzo, usando invece un'unica word di operazione di 16 bit con formato fisso. Per esempio, l'istruzione STOP ha la singola word esadecimale 4E72 come suo codice operativo.

4.5.2 Istruzioni a doppio indirizzo

Quando un'istruzione impiega due operandi, le modalità d'indirizzamento sia per la sorgente che per la destinazione devono essere specificate nella word di operazione. Se ciascuna istruzione a doppio indirizzo codificasse le modalità di indirizzamento in 6 bit ciascuna, come mostrato in precedenza, e specificasse la lunghezza (byte, word o longword) di ciascun operando usando 2 bit, allora occorrerebbero in tutto 14 bit della word di operazione di 16 bit; dunque rimarrebbero soltanto 2 bit per specificare il codice operativo. Poiché sono sempre usati 4 bit per il cod.op., le istruzioni a doppio indirizzo devono avere necessariamente una flessibilità d'indirizzamento limitata, per poter disporre di un insieme completo di istruzioni. Un confronto tra le istruzioni MOVE e ADD illustrerà il metodo adottato dai progettisti dell'MC68020.

L'istruzione MOVE. Il formato dell'istruzione MOVE è illustrato nella Fig. 4.23(a) per MOVE.B, MOVE.L e MOVE.W, in cui i bit [13:12] del cod.op. specificano la lunghezza. Il modo d'indirizzamento della sorgente è specificato come nel caso delle istruzioni a un singolo indirizzo. Invece, il modo d'indirizzamento della destinazione per MOVE inverte la codifica di modo/registro come mostrato in figura. Come esempio, il formato per l'istruzione

```
MOVE.W    D1,D3
```

è illustrato nella Fig. 4.23(b).

L'istruzione ADD. L'istruzione ADD ha il formato illustrato nella Fig. 4.24(a). Esso richiede che l'operando di sorgente o di destinazione sia contenuto in uno dei registri di dati del processore. La forma simbolica dell'istruzione ADD è

```
ADD.X     <EAs>,<Dn>
```

oppure

```
ADD.X     <Dn>,<EAd>
```

con X = B, W o L, come in precedenza. Nel primo caso, Dn specifica la destinazione per il risultato dell'addizione. I bit [8:6] della modalità operativa (Mod.op.) determinano la lunghezza X e specificano la destinazione come Dn. Nella seconda istruzione, la locazione specificata da <EAd> è la destinazione, mentre Dn è la sorgente, per cui la modalità operativa cambia. Un esempio è mostrato nella Fig. 4.24(b).

Molte altre istruzioni a doppio indirizzo vincolano la locazione di sorgente o di destinazione ad essere un registro del processore. Quindi non sono consentite operazioni da memoria a memoria tranne che con l'istruzione MOVE. Questa è dunque la più flessibile tra le istruzioni dell'MC68020 per quanto concerne le modalità d'indirizzamento ammesse.

MOVE Byte

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	Destinazione						Sorgente					
				Registro			Modo			Modo			Registro		

Si noti l'ubicazione di Registro e di Modo.

MOVE Long

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	Destinazione						Sorgente					
				Registro			Modo			Modo			Registro		

Si noti l'ubicazione di Registrc e di Modo.

MOVE Word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	Destinazione						Sorgente					
				Registro			Modo			Modo			Registro		

Si noti l'ubicazione di Registro e di Modo.

(a) *Formati di istruzione.*

	Destinazione		Sorgente	
0011	011	000	000	001
Cod.op.	Registro	Modo	Modo	Registro

(b) MOVE.W D1,D3

Fig. 4.23 *L'istruzione MOVE. (Per gentile concessione di Motorola, Inc.)*

Esempio 4-14

La Fig. 4.25 mostra alcuni esempi delle forme di linguaggio-macchina e di linguaggio assembler per le istruzioni CLR, ADD e MOVE. I valori esadecimali a sinistra dell'istruzione rappresentano il codice di linguaggio-macchina. Qualsiasi valore immediato o indirizzo assoluto è contenuto nelle word di memoria che seguono la word di operazione.

15	14	13	12	11 10 9	8 7 6	5 4 3 2 1 0
1	1	0	1	Registro	Mod.op.	Indirizzo effettivo Modo/Registro

Campi di istruzione:

Campo di registro: specifica uno qualsiasi degli otto registri di dati
Campo Mod.op.:

Byte	Word	Long	Operazione
000	001	010	(< Dn >) + (< ea >) → < Dn >
100	101	110	(< ea >) + (< Dn >) → < ea >

(a) *Formato dell'istruzione ADD.*

	Cod.op.	Registro di dati	Mod.op.	Indirizzo effettivo: Modo	Registro
ADD.W (A1),D1	1101	001	001	010	001
ADD.W D1,(A1)	1101	001	101	010	001

(b) *Codifica dell'istruzione ADD.*

Fig. 4.24 *L'istruzione ADD. (Per gentile concessione di Motorola, Inc.)*

```
                                1.          TTL     FIGURA 4.25
                                2.          LLEN    100
                                3.  *
00010000                        4.          ORG     $1000
                                5.  *
00010000  421D                  6.          CLR.B   (A5)+
00010002  4258                  7.          CLR.W   (A0)+
00010004  42B8 0568             8.          CLR.L   $0568
00010008  4239 00020000         9.          CLR.B   $00020000
                               10.  *
0001000E  D800                 11.          ADD.B   D0,D4
00010010  D378 308E            12.          ADD.W   D1,$308E
00010014  D47C 0030            13.          ADD.W   #$30,D2
                               14.  *
00010018  141D                 15.          MOVE.B  (A5)+,D2
0001001A  3401                 16.          MOVE.W  D1,D2
0001001C  33C1 000300E8        17.          MOVE.W  D1,$300E8
00010022  1CFC 002D            18.          MOVE.B  #'-',(A6)+
00010026  23F9 00020000        19.          MOVE.L  $00022000,$14000
          00014000
                               20.  *
00010030                       21.          END
```

Fig. 4.25 *Esempi di formati d'istruzione per l'MC68020. (Esempio 4-14)*

4.5.3 Compatibilità col codice dell'MC68000

Le istruzioni di 16 bit dell'MC68000 hanno le medesime configurazioni delle corrispondenti istruzioni dell'MC68020. Quindi il linguaggio-macchina o il codice-oggetto per un programma in modo utente scritto per l'MC68000 saranno eseguiti senza modifiche sull'MC68020. Questa compatibilità si estende anche al modo di supervisore, tranne che per certe operazioni che trattano i dati contenuti nello stack di supervisore. Questa lieve incompatibilità è dovuta al fatto che i formati di stack delle informazioni salvate non sono identici per i due processori. Tuttavia, con poche eccezioni, il software sviluppato per la famiglia di 16 bit della Motorola sarà eseguito correttamente su un sistema basato sull'MC68020. I programmi saranno eseguiti più velocemente, poiché l'MC68020 a 32 bit ha un tempo di esecuzione minore per la maggior parte delle istruzioni. Un inconveniente dell'esecuzione sull'MC68020 di programmi di MC68000 non modificati è che l'MC68020 dispone di istruzioni più efficienti per eseguire certe operazioni. Quindi, se tali programmi non vengono riscritti, non potranno sfruttare appieno l'insieme di istruzioni potenziato dell'MC68020.

ESERCIZI

4.5.1 Si scrivano le istruzioni simboliche necessarie per sommare due valori nella memoria e memorizzare il risultato in una terza locazione.

4.5.2 Si supponga che (A1) = $1000 e che ($1000) = $0010 prima dell'esecuzione di ciascuna istruzione elencata di seguito. Si determini l'azione risultante di ciascuna istruzione.

```
(a) CLR.B     $1000
(b) CLR.W     (A1)
(c) MOVE.W    A1,(A1)
(d) MOVE.W    $1000,D1
(e) MOVE.W    #$1000,D1
(f) MOVE.B    (A1),D1
```

Gli indirizzi e i contenuti sono espressi come valori esadecimali.

4.5.3 Si converta ognuno dei seguenti valori di linguaggio-macchina, espressi in esadecimale, nell'equivalente (simbolico) in linguaggio di assemblatore.
(a) 4241
(b) 200B
(c) 103C 002E

4.5.4 Si scriva l'istruzione di linguaggio-macchina per ciascuna delle seguenti istruzioni simboliche:

```
(a) CLR.W     D0
(b) MOVE.L    A0,D0
(c) ADD.B     D0,D5
```

4.5.5 Si consideri il progetto dell'insieme di istruzioni dell'MC68020. Perché alcune istruzioni devono essere limitate nella loro flessibilità d'indirizzamento, rispetto all'istruzione MOVE? Per esempio, l'istruzione ADD richiede che uno degli operandi sia contenuto in un registro di dati.

4.5.6 Si descrivano alcuni fattori che i progettisti devono aver preso in considerazione nella selezione delle istruzioni e delle modalità d'indirizzamento per l'MC68020. (Questo problema è considerato in parecchi articoli elencati nei riferimenti bibliografici relativi a questo capitolo, presentati nell'app. E alla fine del libro.)

4.5.7 Il codice-oggetto dell'MC68020 per i programmi in modo utente è eseguibile su sistemi basati sull'MC68020. Tuttavia, il codice- sorgente di assemblatore non è necessariamente compatibile. In quali circostanze sarebbe desiderabile tale compatibilità del codice-sorgente? In quali circostanze risulta vantaggiosa la compatibilità del codice-oggetto?

4.6 L'MC68020 E L'ORGANIZZAZIONE DELLA MEMORIA

Un diagramma semplificato di un sistema basato sull'MC68020 è illustrato nella Fig. 4.26. L'MC68020 è considerato un processore a indirizzamento di byte, in cui ciascun indirizzo indica una locazione di un byte (8 bit) nella memoria o l'indirizzo di una locazione associata con un'interfaccia. L'intervallo di indirizzi possibili è noto come *spazio d'indirizzamento* o *spazio d'indirizzi* del processore. La Fig. 4.27 illustra tale spazio per le 32 linee d'indirizzo dell'MC68020.

Il progettista di sistema può allocare lo spazio d'indirizzamento per programmi, dati o interfacce di I/O a seconda delle necessità, ma è necessario attenersi a certe convenzioni per prodotti che impiegano l'MC68020. Le 1024 locazioni inferiori sono riservate dal processore MC68020 per essere usate come indirizzi speciali (definiti *vettori* dalla Motorola). Tali indirizzi puntano a routine per il servizio di interruzioni o per l'elaborazione di trappole. Come tali, essi indicano l'indirizzo iniziale di routine del sistema operativo che elaborano le eccezioni nella modalità di

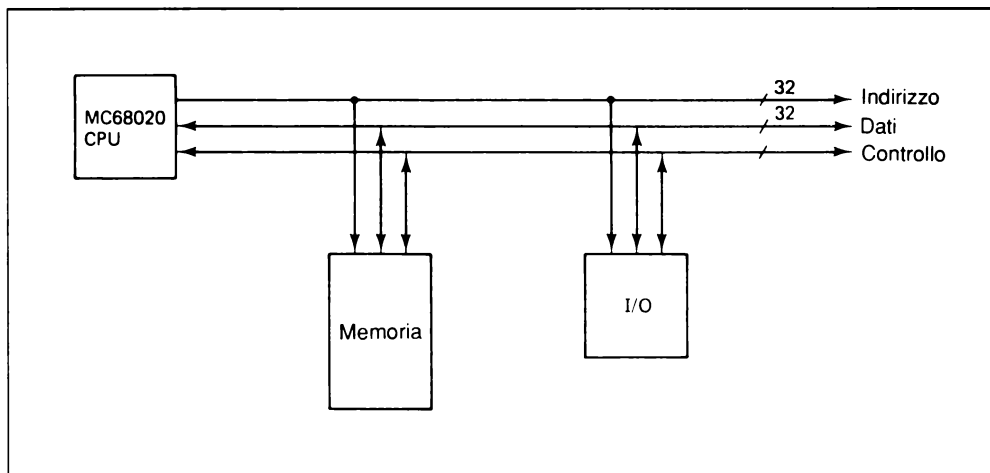
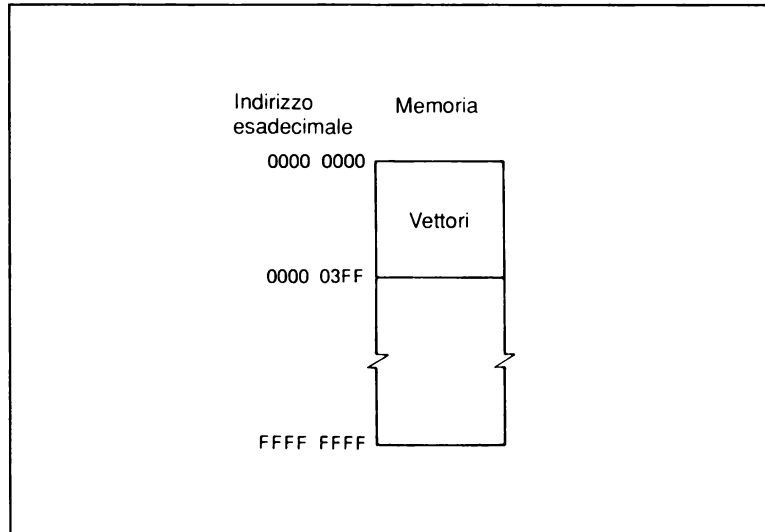



Fig. 4.26 *L'MC68020 e la memoria.*

Fig. 4.27
Spazio d'indirizzi dell'MC68020 per dati di byte.

Indirizzo esadecimale
Memoria
0000 0000
Vettori
0000 03FF
FFFF FFFF

supervisore. In effetti, nei sistemi basati sull'MC68020, possono essere presenti una o più tabelle di vettori d'interruzione.[8]

Poiché l'MC68020 può indirizzare operandi di byte, word o longword, l'organizzazione fisica della memoria in byte — come mostrato nella Fig. 4.27 — potrebbe generare confusione quando vengono indirizzati operandi di word o di longword. Il programmatore dev'essere consapevole della relazione tra l'organizzazione fisica della memoria in byte e la lunghezza dell'operando specificata in un'istruzione. Per un operando lungo un byte, l'indirizzo fisico identifica direttamente il byte indirizzato. Quando in un'istruzione viene specificato un operando di word o di longword, l'indirizzo identifica due o quattro byte nella memoria, rispettivamente.

4.6.1 Organizzazione della memoria e indirizzamento

La configurazione fisica della memoria dell'MC68020 può essere organizzata logicamente in longword, come mostrato nella Fig. 4.28. Questa disposizione corrisponde ad un punto di vista fisico o "hardware" della memoria. Ciascun byte può essere indirizzato singolarmente. Ogni word di memoria occupa due byte ed ha un indirizzo pari. Ogni longword nella memoria occupa quattro byte ed inizia da un indirizzo multiplo di 4. Comunque, questa organizzazione fisica non limita la libertà di un programmatore di far iniziare un *operando* di byte, di word o di longword da un indirizzo qualsiasi, non necessariamente pari. L'unico vincolo cui deve sottostare il programmatore dell'MC68020 è che le istruzioni devono iniziare da indirizzi pari. Pertanto, non è ammessa l'istruzione:

[8] Le tabelle di vettori possono essere anche rilocate nella memoria. Nei progetti standard gli indirizzi della memoria bassa sono riservati ad una tabella di vettori, come si discuterà nel cap. 10.

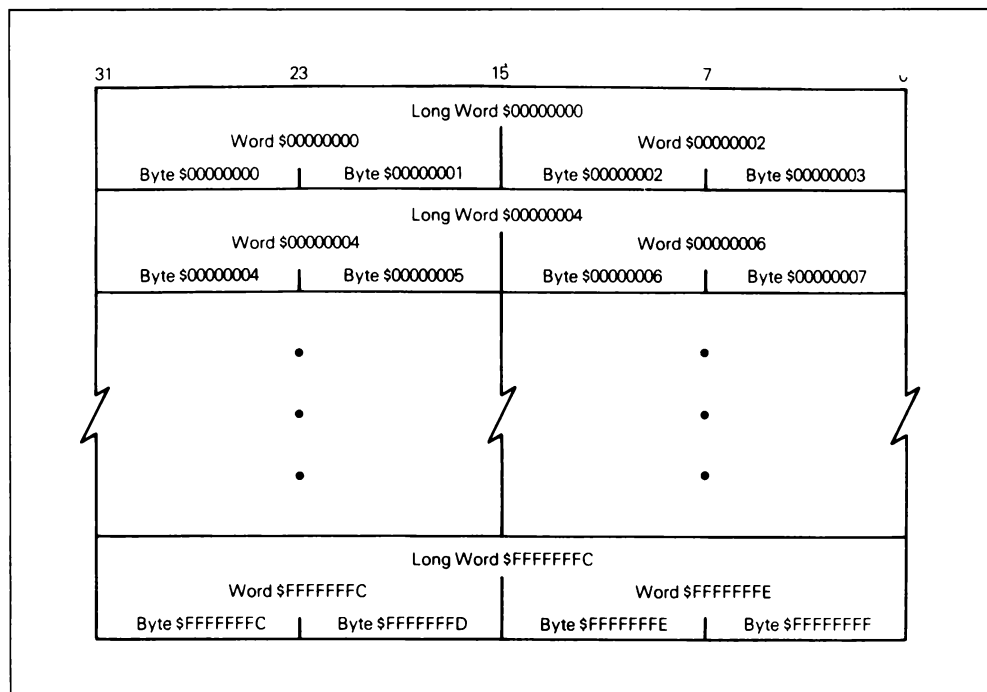



Fig. 4.28 *Organizzazione della memoria per indirizzo di un computer basato sull'MC68020. (Per gentile concessione di Motorola, Inc.)*

```
JMP     1001
```

che tenta d'iniziare l'esecuzione di un nuovo segmento di programma da un indirizzo dispari. Si presuppone che le istruzioni occupino da 1 a 11 word nella memoria, agli indirizzi N, N + 2, N + 4, ..., dove N è un intero pari. L'organizzazione dei valori di dati (operandi) nella memoria sarà considerata nel prossimo paragrafo.

4.6.2 Organizzazione dei dati nella memoria

I valori dei dati o gli indirizzi sono registrati nella memoria come mostrato nella Fig. 4.29. Entro un byte, il bit 0 è quello più a destra, mentre il bit 7 è quello più a sinistra. Per un dato intero, il bit 0 rappresenta la cifra meno significativa in un operando binario di byte, word o longword. La figura mostra gli operandi di word e di longword memorizzati ad un indirizzo di memoria qualsiasi (pari o dispari) ed occupanti rispettivamente il byte o i tre byte successivi. Di solito, gli accessi alla memoria saranno più efficienti se le word e le longword sono memorizzate in locazioni con indirizzi pari. Infatti, la letteratura tecnica della Motorola definisce *allineato* un trasferimento in cui sono impiegati indirizzi pari per fer riferimento ad operandi di word o di longword. Un trasferimento è considerato non allineato quando si fa riferimento ad un operando di word o di longword ad un indirizzo dispari.

Dato intero di byte

7 — 0	7 — 0	7 — 0	7 — 0
Byte n − 1	MSB Byte n LSB	Byte n + 1	Byte n + 2

Indirizzo

Dato intero di word

7 — 0	7 — 0 7 — 0	7 — 0	7 — 0
Byte n − 1	Intero di word	Byte n + 2	Byte n + 3

Indirizzo

Dato intero di longword

7 — 0	7 — 0 7 — 0 7 — 0 7 — 0	7 — 0
Byte n − 1	Intero di longword	Byte n + 4

Indirizzo

Dato codificato in binario impaccato

7 — 0	7 — 4	3 — 0	7 — 0	7 — 0
Byte n − 1	MSD	LSD	Byte n + 1	Byte n + 2

Indirizzo

Dato codificato in binario disimpaccato

7 — 0	7 — 4	3 — 0	7 — 4	3 — 0	7 — 0
Byte n − 1	XX	MSD	XX	LSD	Byte n + 2

Indirizzo

XX = Valore definito dall'utente | MSD = Cifra più significativa LSD = Cifra meno significativa

Fig. 4.29 *Organizzazione dei dati nella memoria per l'MC68020. (Per gentile concessione di Motorola, Inc.)*

Un valore decimale codificato in binario (BCD) è memorizzato con due cifre per byte se è impaccato, mentre un valore non impaccato occupa una word. Per un valore BCD, la cifra meno significativa è sempre quella all'indirizzo di byte più alto nell'operando.

Esempio 4-15

La Tab. 4.7 mostra un certo numero di elementi memorizzati alle locazioni specificate. In ciascun caso, l'indirizzo ed il contenuto sono espressi in esadecimale. Le istruzioni CLR e MOVE richiedono una sola word per la rispettiva word di operazione. L'istruzione MOVE richiede anche un'ulteriore word

Tab. 4.7 *Esempio di contenuti di memoria.*

Indirizzo di memoria	Contenuto (byte)				Significato
	N	N+1	N+2	N+3	
1000	42	83	X	X	CLR.L D3
1006	11	C0	10	00	MOVE.B D0,$1000
100A	00	02	00	F6	INDIRIZZO $200F6
1030	X	10	00	20	LONGWORD (NON ALLIN.)
1034	10	X	X	X	
2000	20	01	X	X	DATO DI BYTE
2008	10	21	X	X	BCD 1021

Note:
1. Tranne che per il valore BCD 1021, tutti i numeri sono in esadecimale.
2. X indica un valore incognito.

per indicare l'indirizzo assoluto corto $1000. L'indirizzo lungo $200F6 è memorizzato come indicato, con la word più significativa che appare per prima nella memoria. Per esempio, un indirizzo di ritorno salvato sullo stack di sistema sarebbe memorizzato in questa maniera. La longword non allineata $1000 2010 alla locazione $1031 è memorizzata come mostrato. L'istruzione

```
MOVE.L    $1031,D1
```

trasferirebbe il valore dalla memoria al registro D1 usando cicli di lettura di 32 bit. La locazione di word $2000 contiene $2001 nella figura, ma si potrebbe indirizzare ogni singolo byte. Quindi l'indirizzo di byte $2000 contiene $20 e l'indirizzo di byte $2001 contiene $01, come mostrato. L'istruzione

```
MOVE.B    $2000,D1
```

produrrebbe (D1)[7:0] = $20. Il trasferimento

```
MOVE.W    $2000,D1
```

risulterebbe in (D1)[15:0] = $2001.

Infine, la locazione $2008 contiene il valore decimale 1021 memorizzato come un numero BCD. Le due cifre meno significative sono memorizzate nella locazione di byte $2009, mentre le due cifre più significative si trovano nella locazione $2008. Le istruzioni dell'MC68020 che operano su numeri BCD a più cifre richiedono questo formato per la memorizzazione di dati BCD.

ESERCIZI

4.6.1

Si determini il numero decimale di byte, word o longword che l'MC6020 può indirizzare.

4.6.2

Si mostri come possono essere memorizzati i seguenti numeri o caratteri se ciascuno inizia dalla locazione esadecimale 1000. I dati e i formati sono i seguenti:
(a) 10203040 (BCD)
(b) 0200 00FC (esadecimale)
(c) 'ABCD' (ASCII)

4.6.3

Il contatore di programma conteneva 0002 $FFF0_{16}$ prima che fosse trasferito nella memoria a partire dalla locazione 1002_{16}. Qual è il contenuto di ciascun byte dell'area di memoria in cui si trova (PC)?

4.6.4

Si mostrino i contenuti delle locazioni di memoria da $10000 a $10007 dopo l'esecuzione dell'istruzione

```
MOVE.L      D1,(A1)
```

con (A1) = $10001 e (D1) = 12 34 56 78 inizialmente. Tutti i valori sono esadecimali. Si tratta di un trasferimento non allineato. Quanti cicli di scrittura sono richiesti?

CAPITOLO 5

IL LINGUAGGIO ASSEMBLER E LE ISTRUZIONI FONDAMENTALI DELL'MC68020

La breve introduzione del cap. 4 al linguaggio-macchina dell'MC68020 dovrebbe aver dato un'idea della complessità insita nella programmazione in linguaggio-macchina. L'estensione dell'insieme di istruzioni, unitamente alla varietà di modi d'indirizzamento possibili per molte istruzioni, dovrebbe precludere qualsiasi tentativo di codifica diretta in linguaggio-macchina, ad eccezione dei programmi più semplici. Nel linguaggio assembler (o assembly), le istruzioni e gli indirizzi sono designati da nomi simbolici, che il programma assemblatore converte nel codice binario appropriato. Per aiutare i programmatori, la Motorola ha definito un linguaggio assembler standard per l'MC68020. Le regole di tale linguaggio specificano i codici mnemonici delle istruzioni, i riferimenti simbolici d'indirizzamento ed il formato di ciascuna istruzione. Queste convenzioni sono generalmente adottate da altri fornitori di assemblatori per l'MC68020. Comunque, le differenze tra i vari assemblatori possono essere stabilite consultando i rispettivi manuali per l'utente.

Questo capitolo inizia con una discussione dello sviluppo del programma, dopodiché saranno presentate le istruzioni del linguaggio assembler per l'MC68020. Saranno poste in evidenza le caratteristiche standard comuni a tutti gli assemblatori. Saranno discusse anche le tecniche di programmazione avanzata che richiedono le capacità più sofisticate di un assemblatore.

In questo capitolo, un numero esadecimale nel testo stesso sarà preceduto dal simbolo "$"; per il resto, i valori numerici nel testo saranno espressi in decimale. Tuttavia i listati di assemblatore e le informazioni di uscita di sessioni di monitor impiegano valori esadecimali per gli indirizzi delle locazioni di memoria e per i relativi contenuti, ma questi programmi non utilizzano alcun prefisso per indicare la notazione esadecimale. Soltanto le espressioni del linguaggio assembler create dal programmatore richiedono la forma $NNN per indicare un numero esadecimale come valore immediato o come indirizzo di memoria di un operando.

Quando un registro d'indirizzo è impiegato come locazione di destinazione in un'istruzione di un esempio, le varianti delle istruzioni ADD, MOVE e SUB diverranno ADDA, MOVA e SUBA (il suffisso "A" aggiunta è l'iniziale di *address*, "indirizzo"). Le varianti delle istruzioni per il modo immediato (da 16 a 32 bit) e per quello immediato rapido (3 bit) saranno ADDI, ADDQ, SUBI, SUBQ, e così via (Il suffisso "I" è l'iniziale di "immediato"; il suffisso "Q" è l'iniziale di *quick*, "rapido"). La maggior parte degli assemblatori riconosce le varianti delle istruzioni anche se il suffisso non è definito esplicitamente; tuttavia, per maggior chiarezza, i programmi usati come esempi in questo capitolo impiegheranno le varianti col suffisso. Per comodità, l'insieme delle istruzioni in linguaggio assembler per l'MC68020 è riassunto nell'app. C. Le varianti delle istruzioni aritmetiche saranno presentate nel cap. 7.

5.1 SVILUPPO DEL SOFTWARE

Lo sviluppo del software consiste dell'analisi del problema, della progettazione dell'hardware e della codifica del programma, seguiti dal debugging e dai test. In ciascuno stadio, dovrebbe essere prodotta una documentazione appropriata. Le attività di programmazione sono mostrate in forma semplificata nella Fig. 5.1, che evidenzia la natura ciclica o iterativa del processo. Il programma di editor è usato per creare un programma *sorgente* in linguaggio assembler che sarà tradotto dall'assemblatore.[1] In questo stadio di sviluppo, viene utilizzato il *listato* prodotto dall'assemblatore per trovare gli errori nel programma sorgente. Se nessun errore viene rivelato dall'assemblatore, allora il listato conterrà il programma in linguaggio assembler e l'equivalente in linguaggio-macchina; in caso contrario, qualunque errore sarà chiaramente indicato.

Quando un programma è esente da errori di assemblaggio, sarà prodotto un *programma-oggetto*, che sarà caricato nella memoria della macchina di destinazione ("target") per essere eseguito. In questa discussione semplificata, il programma-oggetto è un programma in linguaggio-macchina.[2] L'esecuzione del codice viene controllata in fase di debugging da un programma denominato *debugger*. Questo programma permette all'utente di procedere passo-passo nell'esecuzione delle istruzioni e di visualizzare i risultati intermedi allorché ogni istruzione viene completata. Gli errori nella progettazione del programma possono essere rivelati in questo stadio. Per correggere gli errori, il programma-sorgente dovrà essere rieditato e riassemblato.

---

[1] I dettagli impliciti nell'esecuzione del software di sviluppo (editor, assemblatore, debugger) variano notevolmente a seconda dei sistemi. Inoltre, i programmi sorgente e oggetto sono normalmente memorizzati in file sul disco del sistema di sviluppo. Il manuale per l'utente o il manuale del sistema operativo per un particolare sistema descriveranno il procedimento richiesto per creare, memorizzare ed eseguire i programmi.

[2] In pratica, il programma-oggetto può richiedere l'elaborazione da parte di un altro programma, denominato *linkage editor* (editor di collegamento), prima di essere caricato nella memoria per l'esecuzione. La distinzione tra le operazioni di assemblaggio, caricamento ed esecuzione è discussa in vari riferimenti bibliografici relativi a questo capitolo, riportati nell'app. E.

Fig. 5.1
Programmazione di un micro-computer.

Inizio

Caricamento del programma di editing

Digitazione del program. sorgente e correzione

Programma sorgente

Assemblaggio

Errori?

Sì

No

Programma oggetto

Caricamento del programma oggetto

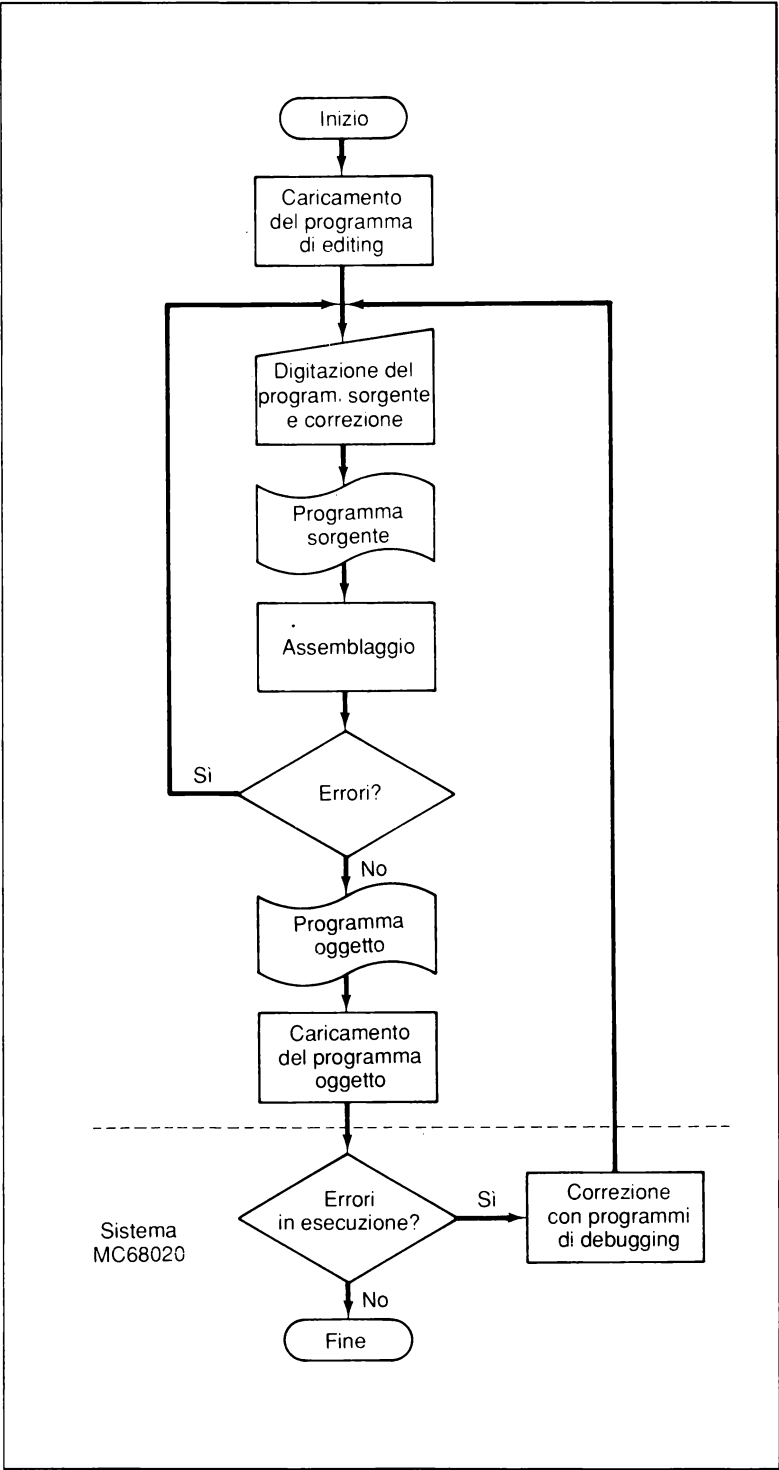
Sistema MC68020

Errori in esecuzione?

Sì

Correzione con programmi di debugging

No

Fine

5.1.1 L'assemblatore ed il listato

Come notato in precedenza, il processo di assemblaggio esamina ciascuna espressione (*statement*) del programma-sorgente per verificare che non ci siano errori. Ogni espressione può essere un'istruzione dell'MC68020, una direttiva dell'assemblatore, o un commento. Un'istruzione simbolica quale:

```
ADD.W    D1,D2
```

diviene un'istruzione eseguibile in linguaggio-macchina. Il codice mnemonico ADD, la dimensione dell'operando e gli indirizzi degli operandi vengono riconosciuti dall'assemblatore e convertiti in codice-macchina binario. D'altro canto, le direttive dell'assemblatore sono istruzioni per l'assemblatore, non per la CPU. Per esempio, la direttiva di origine (ORG) specifica il punto della memoria a partire dal quale dovrà essere caricato il programma. Quindi, la direttiva

```
ORG      $10000
```

indica che il programma dev'essere caricato a partire dalla locazione decimale $10000. Nel programma-sorgente possono comparire anche dei commenti per la comodità del programmatore. Questi commenti vengono ignorati dall'assemblatore ma saranno stampati sul listato.

La Fig. 5.2 mostra un listato tipico dell'MC68020, nel medesimo formato con cui i listati di esempio saranno presentati in questo capitolo. La terza colonna è il numero decimale della riga. La prima colonna è il valore esadecimale del contatore di locazione in ciascuna istruzione. Il contatore di locazione tiene nota delle locazioni delle istruzioni durante l'assemblaggio, così come fa il contatore di programma durante l'esecuzione del programma. Se il programma mostrato fosse stato caricato a partire dalla locazione $10000, il PC cambierebbe come il contatore di locazione nella Fig. 5.2. La seconda colonna rappresenta la conversione in linguaggio-macchina, che mostra la word di operazione per ogni istruzione, seguita dal valore di qualsiasi word di estensione richiesta per l'istruzione. In questa colonna appare anche qualunque valore assegnato da una direttiva di assemblatore, come l'espressione di definizione di una costante (*Define Constant:* DC) mostrata nella figura. Il numero decimale della riga è seguito alla sua destra dall'espressione del programma-sorgente che l'ha generata. Negli esempi di questo libro, i commenti in una riga sono preceduti dal punto e virgola (;) facoltativo; un'intera riga può essere definita come un commento se il suo primo carattere è un asterisco (*).

Il semplice programma della Fig. 5.2 esegue la somma di quattro interi di 16 bit nelle locazioni $20000, $20002, $20004, $20006. Il risultato viene memorizzato in (D1)[15:0]. Le tre direttive LLEN, ORG e END definiscono rispettivamente la larghezza del listato, e l'origine e la fine del programma. La direttiva TTL (titolo) è facoltativa e serve a identificare il programma per la comodità del programmatore. INIT e LOOP sono etichette associate ad una particolare riga, cosicché tale riga può essere indicata simbolicamente da qualsiasi altro punto nel programma. I valori delle etichette sono assegnati dal contatore di locazione.

```
                                  1.         TTL      FIGURA 5.2
                                  2.         LLEN     100             ;LUNGHEZZA DELLA RIGA
00010000                          3.         ORG      $10000          ;ORIGINE NELLA MEMORIA
                                  4.   *
                                  5.   * SOMMA 4 NUMERI DI 16 BIT MEMORIZZATI NELLE LOCAZIONI
                                  6.   * DA $20000 A £20006. RIPORTA LA SOMMA IN D1[15:0].
                                  7.   *
00010000  7200                    8.   INIT  MOVE.L   #0,D1           ;SOMMA ZERO
00010002  227C 00020000           9.         MOVEA.L  #$20000,A1      ;INDIRIZZO DEL PRIMO NUMERO
00010008  7404                   10.         MOVE.L   #4,D2           ;PONE IL CONTATORE A 4
                                 11.   *
                                 12.   * DEFINISCE IL CICLO PER LA SOMMA DEI VALORI
                                 13.   *
0001000A  D259                   14.   LOOP  ADD.W    (A1)+,D1        ;SOMMA I NUMERI
0001000C  5342                   15.         SUB.W    #1,D2           ;DECREMENTA IL CONTATORE
0001000E  66 FA                  16.         BNE      LOOP            ;FINCHE' (D2)=0
                                 17.   *
00010010  4E4F                   18.         TRAP     #15             ;RITORNA AL MONITOR
00010012  0063                   19.         DC.W     $0063
00010014                         20.         END
```

Fig. 5.2 *Un tipico listato di programma in linguaggio assembler.*

Nel programma, INIT indica la locazione della prima istruzione. LOOP ("ciclo") definisce l'inizio di una sequenza di istruzioni che viene eseguita ripetutamente per quattro volte per sommare i quattro valori. Questa iterazione o ciclo termina allorché il valore in D2 raggiunge lo zero.

L'ultima istruzione, TRAP #15, serve a restituire il controllo al programma di monitor. Questo è il modo tipico di trasferire il controllo da un programma di utente ad un sistema operativo o ad un programma di monitor della Motorola. Una costante che viene definita come facente parte dell'istruzione TRAP da DC.W $0063 (*Define Constant:* definizione di costante) indica al monitor che il programma di utente ha completato l'esecuzione quando viene eseguita l'istruzione TRAP. Sono disponibili varie altre opzioni, come sarà descritto in seguito. L'istruzione END è una direttiva all'assemblatore, che definisce la fine del programma da assemblare. Essa dev'essere l'ultima direttiva nel programma.

Il particolare assemblatore impiegato in quest'esempio è un cross-assemblatore prodotto dalla società Quelo di Seattle, Washington. L'assemblatore riconosce tutte le istruzioni in linguaggio assembler dell'MC68020 e converte il programma assembler in un modulo-oggetto. Viene prodotto anche un listato come quello mostrato nella Fig. 5.2. Tuttavia, il modulo-oggetto non è pronto per l'esecuzione, ma dev'essere dapprima elaborato sul computer host da un programma di collegamento (*linker*) e poi dal programma di debugger del sistema basato sull'MC68020. Il sistema impiegato per la maggior parte degli esempi in questo libro è il computer su piastra singola MVME133, già descritto nel cap. 1. Il programma debugger utilizzato fa parte in realtà del programma di monitor memorizzato in una ROM sulla piastra del modulo MVME133.

Quando s'impiegano le tecniche di cross-assemblaggio discusse nel par. 1.2, l'esecuzione dell'assemblatore avviene su un computer host. I programmi assemblati e collegati devono successivamente essere caricati nella memoria del sistema target; la piastra MVME133, in questo caso. Queste tecniche e l'utilizzazione

del monitor BUG dell'MVME133 saranno descritte nei prossimi due paragrafi. L'assemblatore residente disponibile nei computer della Motorola sarà descritto nel par. 5.1.4.

5.1.2 Cross-assemblaggio e collegamento

La Fig. 5.3 illustra i sistemi del computer host e il software utilizzati dall'autore per lo sviluppo dei programmi con cross-assemblatori. Per lo sviluppo del software sono stati utilizzati il modello 11/780 del VAX (*Virtual Address Estension:* estensione di indirizzo virtuale) prodotto dalla Digital Equipment Corporation o il PC (*Personal Computer*) dell'IBM. Una volta che il programma in linguaggio assembler è corretto e pronto per essere eseguito, il modulo di caricamento può essere trasferito al sistema basato sull'MC68020 per l'esecuzione e il debugging. In alternativa, l'esecuzione può essere simulata mediante un programma simulatore sull'uno o l'altro dei computer host, come descritto nel par. 1.2.2. Tutto il software descritto qui per lo sviluppo dei programmi è stato fornito dalla Quelo per entrambi i computer host. Questi programmi sono eseguiti sotto il controllo dell'appropriato sistema operativo del computer host utilizzato.

Il primo passo consiste nella creazione del programma-sorgente e nella sua conversione mediante l'assemblatore in un modulo- oggetto *rilocabile.* Questo modulo-oggetto in genere non contiene indirizzi assoluti che fanno riferimento alla memoria di sistema dell'MC68020, bensì indirizzi rilocabili che saranno assegnati dal programma *linkage editor* (editor di collegamento). Questo programma serve a combinare vari moduli-oggetto in un unico *modulo di caricamento.* Il file di collegamento elenca i nomi ed altre informazioni relative ai moduli da collegare. Tutti i programmi sono memorizzati in file su disco nell'unità a disco del computer host.

Nel trasferimento del modulo di caricamento dal computer host al sistema dell'MC68020 il modulo di caricamento viene trattato come un file di testo in ASCII. Il programma di monitor nella ROM della piastra MVME133, sotto il controllo dell'operatore del computer, richiede il trasferimento dall'host. Il monitor converte il modulo-oggetto trasmesso dalla forma di S-record (ASCII) in istruzioni di linguaggio-macchina, dopodiché carica nella memoria le istruzioni eseguibili. Il formato di S-record è impiegato dalla Motorola per il trasferimento di programmi e dati tra computer. Il computer host deve avere soltanto il programma di trasferimento del file per ottemperare alla richiesta del monitor.[3] Una volta che il programma in linguaggio-macchina è stato trasferito nella memoria, viene utilizzato il monitor per eseguire il programma e svolgere le funzioni di debugging, come sarà descritto nel prossimo paragrafo.

[3] Se un programma di trasferimento di file non fosse disponibile, se ne potrebbe scrivere facilmente uno per programmare il chip periferico che controlla le linee di segnale del computer host per il trasferimento di dati seriali. Ciò è stato necessario per il PC IBM nel caso dell'autore. Tali chip saranno discussi nel cap. 13.

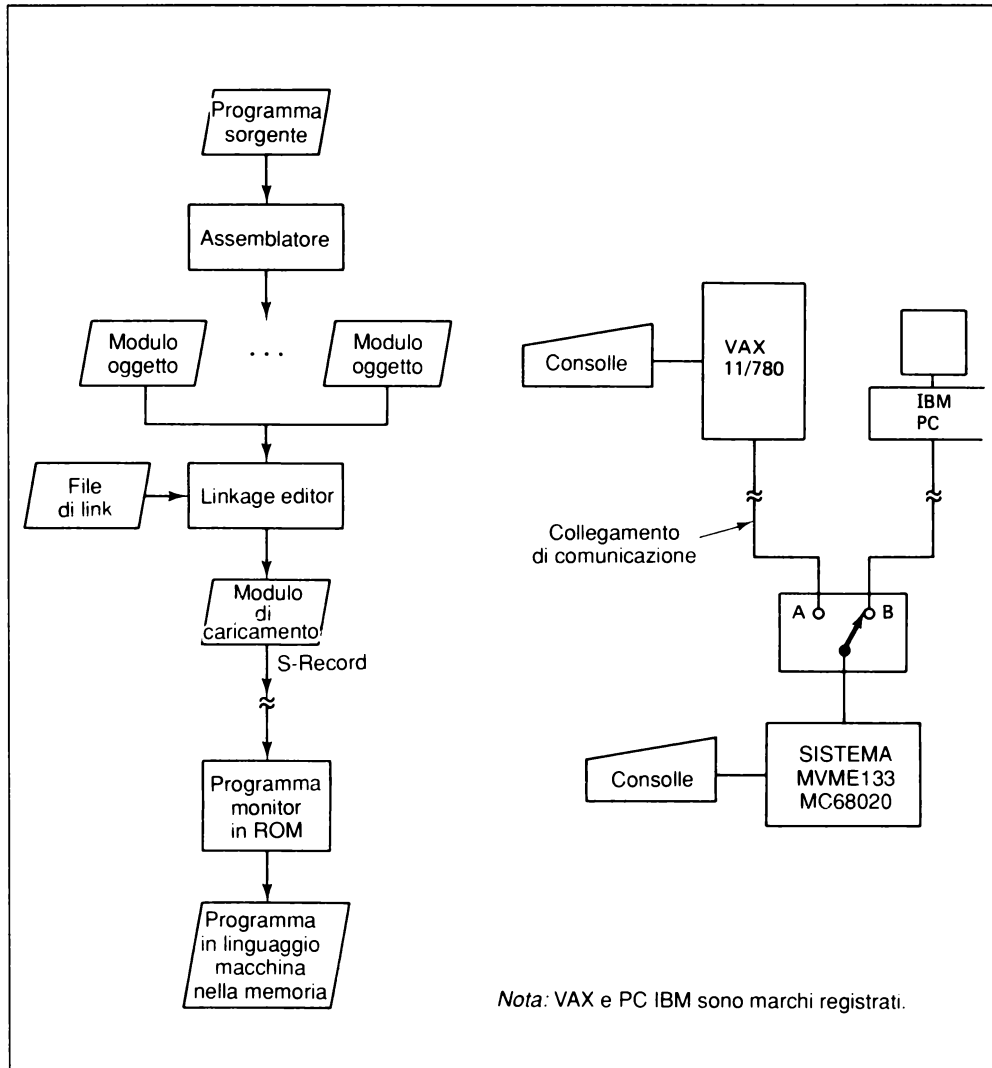



Fig. 5.3 *L'hardware e il software richiesti nello sviluppo di un programma.*

Esempio 5-1

Il modulo di caricamento prodotto dal cross-software non è eseguibile direttamente su un sistema basato sull'MC68020. Questo modulo viene memorizzato in un file su disco nell'unità a disco del computer host. Il file contiene effettivamente del testo ASCII, in un formato speciale definito come *S-record* dalla Motorola. Tale formato è stato creato per consentire la trasmissione di file tra sistemi di computer attraverso linee di comunicazione seriale. Il formato generale è illustrato nella Fig. 5.4(a), mentre un esempio specifico per il programma in linguaggio assembler della Fig. 5.2 è mostrato nella Fig. 5.4(b).

Tipo	Lunghezza del record	Indirizzo	Codice/Dati	Check Sum

Tipo: S0-S9; 2 caratteri.
Lunghezza: valore esadecimale; è il numero di byte (coppie di caratteri) nel record, inclusi l'indirizzo e la somma di verifica; 2 caratteri.
Indirizzo: valore esadecimale: è l'indirizzo iniziale nella memoria; 4 (S1), 6 (S2) o 8 (S3) caratteri.
Codice/Dati: dati o istruzioni.
Check Sum: (somma di verifica); è il byte meno significativo della somma in complemento a 1 dei valori nel record, escluso il tipo; 2 caratteri.

(a) *Definizione del formato.*

```
S00B000066352D352E68657884
S21801000072002270C000200007404D259534266FA4E4F00633C
S9030000FC
```

(b) *Esempio di modulo-oggetto nel formato di S-record.*

Fig. 5.4 *Formato di S-record.*

Il record S0 indica il primo record di una serie di S-record. Ognuno dei record successivi, (S1), (S2) o (S3), contiene il tipo, la lunghezza esadecimale del record in byte, l'indirizzo iniziale esadecimale in cui dovrà essere caricato il segmento di programma, e infine le istruzioni o i dati. La lunghezza è il numero di coppie di caratteri che seguono la specificazione di lunghezza. Ci sono 24 (cioè \$18) byte o coppie di caratteri nel record S2 di Fig. 5.4(b). Questi valori, senza i caratteri di somma di controllo (*checksum*) devono essere registrati nella memoria a partire dall'indirizzo \$10000. Gli ultimi due caratteri rappresentano una somma di verifica, che serve appunto a verificare la correttezza del record. Un record di terminazione (S9) indica la fine del modulo-oggetto. Si noti che la somma di verifica \$FC rappresenta il complemento a 1 della somma dei valori esadecimali (\$03) nel record S9, esclusa la stessa specificazione del tipo.

5.1.3 Il programma monitor

Il monitor della Motorola, utilizzato per l'esecuzione e il debugging della maggior parte degli esempi di programma illustrati in questo libro, è designato come MVME133BUG o con la sigla abbreviata 133BUG. Uno dei suoi scopi principali è quello di accettare i comandì dall'operatore tramite il terminale, che consiste di una tastiera e di un'unità di visualizzazione. Questi comandi consentono all'operatore di selezionare funzioni come le seguenti:

(a) Avviare i trasferimenti tra la memoria e l'unità a disco.

(b) Eseguire test diagnostici sui componenti hardware del sistema.

(c) Servire nel debugging del programma.

Questo monitor è residente in una memoria a sola lettura (ROM) e richiede 64 kilobyte di memoria ad accesso casuale addizionale. La sua capacità consente ad un computer su singola piastra MVME133 di fungere da piastra di CPU in un sistema basato sull'MC68020 con una o più unità a disco. Nei sistemi privi di unità a disco e di sistema operativo, il 133BUG è usato soprattutto per la valutazione e il debugging del programma.

Caratteristiche di sistema del monitor. All'accensione del sistema di computer, il programma monitor nella ROM è il solo software disponibile per il sistema. Il monitor 133BUG inizia automaticamente ad essere eseguito e svolge vari test diagnostici sul sistema. Se il computer non presenta malfunzionamenti, il monitor richiede un comando all'operatore, visualizzando sull'unità video del terminale il seguente messaggio di richiesta:

133BUG>

dopodiché attende che l'operatore inserisca un comando valido. Un comando di esempio è BO (*Bootstrap Operating system:* carica il sistema operativo). Se il computer ha un sistema operativo memorizzato nella propria unità a disco, questo comando ne trasferirà delle porzioni dall'unità a disco alla memoria, dopodiché cederà il controllo al sistema operativo. Nei computer provvisti di un sistema operativo, il monitor non è utilizzato per le normali operazioni, poiché in questo caso è il sistema operativo a svolgere la funzione di programma supervisore per il computer. Questo caso sarà descritto nel prossimo paragrafo.

Oltre a permettere all'operatore di svolgere operazioni che riguardano un'unità a disco, il monitor contiene alcune routine che consentono di eseguire vari test diagnostici sull'hardware del computer stesso. L'operatore può selezionare i test diagnostici per il chip della CPU, per la memoria o per qualsiasi coprocessore eventualmente incluso nel sistema. Se qualche dispositivo non superasse i test, allora sarebbe visualizzato sul terminale un messaggio indicante la natura del difetto. In questa circostanza, l'operatore può intraprendere la necessaria azione di rimedio. Per esempio, egli potrebbe sostituire un modulo di memoria, se la memoria non avesse superato i test diagnostici.

Il monitor 133BUG è stato progettato per essere il monitor fondamentale in un sistema di computer complesso. Normalmente, la procedura d'inizializzazione consiste nell'eccensione del sistema da parte dell'operatore e nel caricamento immediato (tramite il comando BO) del sistema operativo. Soltanto nel caso in cui l'operatore sospettasse la presenza di un guasto in qualche punto del sistema, dovranno essere inviati i comandi di esecuzione degli appropriati test diagnostici.

Se non è presente alcun sistema operativo, sarà usato il monitor per il caricamento e il debugging del programma.

Sviluppo e debugging del programma. In generale, qualsiasi programma monitor consentirà all'operatore di caricare un programma, inizializzare le locazioni della memoria ed i registri del processore ed eseguire il programma. La sessione di monitor viene svolta in maniera interattiva tramite un terminale di operatore. Al terminale, i valori possono essere modificati durante l'esecuzione del programma (nei *breakpoint*) ai fini del test. Inoltre possono essere visualizzati i contenuti di locazioni e di registri. Se vengono scoperti degli errori, di solito il programma viene riassemblato, collegato e caricato per un'altra sessione di debugging. Il 133BUG discusso qui è tipico di programmi di monitor per i moderni sistemi di computer. Tuttavia, le caratteristiche di un particolare monitor dipendono completamente dal computer utilizzato, per cui è necessario consultare il manuale per l'utente del monitor per comprenderne l'uso.

La Tab. 5.1 elenca alcuni comandi per il monitor 133BUG. Il messaggio di richiesta (*"prompt"*) che il monitor visualizza sullo schermo

133BUG>

indica che il monitor è pronto ad accettare comandi. L'operatore inserisce allora il comando di due lettere, seguito da eventuali parametri, dopodiché pigia il tasto di ritorno-carrello (*Carriage Return:* CR) per inviare il comando. Qualunque valore visualizzato dal monitor in risposta ad un comando non sarà preceduto dal prompt negli esempi di questo capitolo; inoltre, tutti gli indirizzi e i contenuti di locazioni della memoria saranno espressi in esadecimale, a meno che non sia indicato altrimenti. Per tale motivo, i valori esadecimali inseriti dall'operatore non adottano la designazione "$" che è richiesta da una notazione in linguaggio assembler. I registri del processore saranno designati dai rispettivi nomi simbolici: A0, A1, ..., A6 per i registri d'indirizzo, e D0, D1, ..., D7 per i registri di dati. Gli altri registri d'interesse immediato sono PC (*Program Counter:* contatore di programma) e CCR (*Condition Code Register:* registro dei codici di condizione).

I comandi di monitor nella Tab. 5.1 sono suddivisi in quelli che consentono di comunicare con un computer host e quelli che servono per il debugging del programma. Nella prima categoria, TM (*Transparent Mode*) e LO (*LOad*) sono utilizzati rispettivamente per stabilire la comunicazione col computer host e per trasferire un file dall'host al computer target. Il nome del file da trasferire fa parte del comando specificato in LO per l'host. Per il debugging, il contenuto di un registro o di una locazione di memoria può essere inizializzato, modificato, o visualizzato dal comando appropriato nella Tab. 5.1. Un certo numero di altri comandi controllano l'esecuzione di un programma. Per esempio, i comandi T (*Trace*) e TC (*Trace on Change of control flow*) definiscono le opzioni di traccia per la CPU. Nelle modalità di traccia, vengono eseguite le istruzioni "tracciate", dopodiché il controllo è restituito al programma monitor. Quest'ultimo visualizzerà di solito i contenuti dei registri della CPU dopo il tracciamento.

Tab. 5.1 *Comandi di 133BUG.*

FORMATO GENERALE	
133BUG>	< comandi inseriti dall'operatore> (CR)
COMUNICAZIONI COL COMPUTER HOST	
L0; X= <comando a host>	*Load:* carica S-record da host
TM	*Transparent Mode:* modo trasparente (stabilisce le comunicazioni con l'host)
INIZIALIZZAZIONE DI REGISTRI O DELLA MEMORIA	
MM <indirizzo> [;DI]	*Memory Modify*: modifica memoria (sequenziale)
MS <indirizzo> <valore>	*Memory Set:* definisci memoria (una locazione)
RM <registro>	*Register Modify:* modifica registro
VISUALIZZAZIONE DI REGISTRI O DELLA MEMORIA	
MD <indirizzo iniziale> [;DI]	*Memory Display:* visualizza memoria [:disassembla]
RD <registri>	*Register Display:* visualizza registri
ESECUZIONE E TRACCIA	
BR <indirizzo>	*Breakpoint:* inserisci un breakpoint
GO	*Go:* vai a (esegui)
GT <indirizzo>	*Go Till:* esegui fino a <indirizzo>
T	*Trace:* procedi passo-passo (traccia)
TC	*Trace on Change of control flow:* traccia sul cambiamento del flusso di controllo

Note:
1. <> indica qualsiasi selezione valida.
2. Tutti i valori per i dati e gli indirizzi sono esadecimali.
3. I registri sono selezionati dai rispettivi codici mnemonici in linguaggio assembler.
4. [;DI] indica un valore opzionale; in questo caso, "assembla" o "disassembla" (DI).

Il monitor suddetto dispone inoltre di un assemblatore di una sola riga ("*one-line*"), per convertire in linguaggio-macchina una singola istruzione del linguaggio assembler all'indirizzo specificato. Essa è un'opzione per il comando MM (*Memory Modify*). Questo assemblatore di una sola riga è di utilità limitata nello sviluppo di un programma. Per esempio, non sono ammesse etichette come nella Fig. 5.2.

Il processo inverso all'assemblaggio è noto come *disassemblaggio*. Quando vengono visualizzate e disassemblate le istruzioni in linguaggio-macchina contenute in locazioni di memoria, il monitor 133BUG visualizzerà in forma mnemonica le corrispondenti istruzioni in linguaggio assembler. Esse possono essere usate per verificare la correttezza del programma nella memoria senza convertire il codice di linguaggio-macchina quando viene usato il comando MD (*Memory Display*).

Esempio 5-2

L'uso del monitor 133BUG per il caricamento, l'esecuzione e il debugging del programma è illustrato nella Fig. 5.5(b). Un listato in linguaggio assembler del programma è stato incluso per comodità nella Fig. 5.5(a). Il suo modulo-oggetto è memorizzato nell'unità a disco del computer host, nel file F5-5.Hex. Ciò è indicato anche nel primo S-record quando viene letto come insieme di caratteri ASCII. Questo file di S-record è stato creato usando i programmi di cross-assemblatore e Linker (collegatore) della Quelo. Prima che il modulo-oggetto possa essere caricato nella memoria, è necessario connetterlo (*log on*) al computer host per stabilire il collegamento di comunicazione. Per il monitor 133BUG, ciò avviene inviando il comando TM (*Transparent Mode*). Di conseguenza, il computer host riconosce il modulo MVME133 come se fosse semplicemente un altro terminale. Questa parte della sessione non è mostrata nella figura. Allorché tale comunicazione con l'host sarà stata stabilita, il modulo-oggetto potrà essere caricato richiamando di nuovo il monitor col comando LO (*LOad*).

Nella figura, il testo del comando LO dopo il segno "=" viene trasmesso all'host. Questo risponde trasmettendo il file di S-record, che viene visualizzato sul terminale del sistema basato sull'MC68020. Il programma viene anche caricato dal monitor nella memoria a partire dalla locazione $10000. È ora possibile sconnettere l'MVME133 dal computer host (*log off*), poiché questo non è più richiesto nella sessione di debugging.

Questo semplice programma somma quattro numeri di 16 bit nelle locazioni di memoria $20000-$20006 ed accumula la somma in (D1)[15:0]. L'indirizzo del primo numero viene caricato in A1 dall'istruzione MOVEA, una variante dell'istruzione MOVE. Per collaudare il programma, quattro valori di test vengono caricati nella memoria come mostrato nella Fig. 5.5(b), servendosi del comando MM (*Memory Modify*). Sono usati i valori 1, 2, 3 e 4. L'ultimo valore è seguito da un punto per indicare al monitor che non devono essere inizializzate ulteriori locazioni. I valori sono quindi visualizzati (MD) per verificarne la correttezza. Poi il contatore di programma viene inizializzato a $10000.

Prima dell'esecuzione, sono visualizzati i contenuti dei registri. Durante la sessione di debugging, sono visualizzati soltanto i registri d'interesse (PC, D1, D2 e A1). A tal fine, è stato usato il comando RD come mostrato, con le specificazioni appropriate per la selezione dei registri da visualizzare.

```
                              1.         TTL     FIGURE 5.5
                              2.         LLEN    100             ;LINE LENGTH
00010000                      3.         ORG     $10000          ;ORIGIN IN MEMORY
                              4.  *
                              5.  * ADD FOUR 16-BIT NUMBERS STORED IN LOCATIONS $20000
                              6.  *    THROUGH $20006.  RETURN THE SUM IN D1[15:0].
                              7.  *
00010000  7200                8.  INIT   MOVE.L  #0,D1           ;ZERO SUM
00010002  227C 00020000       9.         MOVEA.L #$20000,A1      ;ADDR OF FIRST NUMBER
00010008  7404               10.         MOVE.L  #4,D2           ;SET COUNTER TO 4
                             11.  *
                             12.  * DEFINE LOOP TO ADD VALUES
                             13.  *
0001000A  D259               14.  LOOP   ADD.W   (A1)+,D1        ;SUM NUMBERS
0001000C  5342               15.         SUB.W   #1,D2           ;DECREMENT COUNTER
0001000E  66 FA              16.         BNE     LOOP            ;TILL (D2)=0
                             17.  *
00010010  4E4F               18.         TRAP    #15             ;RETURN TO MONITOR
00010012  0063               19.         DC.W    $0063
00010014                     20.         END
```

(a) *Listato del programma in linguaggio assembler.*

```
133Bug>LO:X=T F5-5.HEX
T F5-5.HEX
S00B000066352D352E68657884
S21801000007200227C000200007404D259534266FA4E4F00633C
S9030000FC
133Bug>MD 10000:DI
00010000 7200                MOVEQ.L     #$0,D1
00010002 227C0002 0000       MOVEA.L     #$20000,A1
00010008 7404                MOVEQ.L     #$4,D2
0001000A D259                ADD.W       (A1)+,D1
0001000C 5342                SUBQ.W      #$1,D2
0001000E 66FA                BNE.B       $1000A
00010010 4E4F0063            SYSCALL     .RETURN
00010014 FFFF                DC.W        $FFFF
133Bug>MM 20000
00020000 0000? 0001
00020002 0000? 0002
00020004 FFFF? 0003
00020006 FFFF? 0004.
133Bug>MD 20000
00020000 0001  0002  0003  0004   0000  0000  FFFF  FFFF     ........
133Bug>RM PC
PC    =00000000 ? 10000.
133Bug>RD
PC    =00010000 SR    =2700=TR:OFF_S._7_.....
USP   =00003830 MSP   =00003C18 ISP* =00004000 VBR  =00000000
SFC   =0=XX     DFC   =0=XX    CACR =0=..      CAAR =00000000
D0    =00000000 D1    =00000000 D2   =00000000 D3   =00000000
D4    =00000000 D5    =00000000 D6   =00000000 D7   =00000000
A0    =00000000 A1    =00000000 A2   =00000000 A3   =00000000
A4    =00000000 A5    =00000000 A6   =00000000 A7   =00004000
00010000 7200                MOVEQ.L     #$0.D1
133Bug>RD=PC/D1/D2/A1
PC    =00010000 D1    =00000000 D2   =00000000 A1   =00000000
00010000 7200                MOVEQ.L     #$0,D1
133Bug>GT 10010
Effective address: 00010010
Effective address: 00010000
At Breakpoint
PC    =00010010 D1    =0000000A D2   =00000000 A1   =00020008
00010010 4E4F0063            SYSCALL     .RETURN
```

(b) *Monitor.*

Fig. 5.5 *Una sessione di monitor.*

Successivamente il comando GT avvia l'esecuzione, che procede dalla locazione iniziale fino all'istruzione in $10010. La visualizzazione finale nella Fig. 5.5(b) mostra che il valore in D1 è in effetti la somma esadecimale dei valori degli addendi. Il contatore (D2) è stato decrementato a zero, mentre il registro d'indirizzo è stato incrementato di 2 ogni volta attraverso il ciclo per puntare all'operando successivo nella memoria. Il comando GO causa l'esecuzione dell'istruzione TRAP #15, dopodiché il controllo viene restituito al monitor.

Il debugging potrebbe continuare con valori differenti nella memoria e col (PC) ripristinato a $10000. Un test più esteso rivelerebbe una pecca nel programma, poiché la somma nel registro D1 non viene esaminata per verificare che non produca un superamento della lunghezza (16 bit) del registro impiegato. Le tecniche di test per la situazione considerata qui saranno discusse nei capp. 6 e 7.

5.1.4 L'assemblatore residente

L'assemblatore residente (*Resident Assembler*) della Motorola opera sotto il controllo del sistema operativo della Motorola. Tale programma converte le istruzioni-sorgente scritte in linguaggio assembler nel codice-oggetto rilocabile. Uno o più moduli-oggetto possono essere collegati dal programma *linkage editor* della Motorola e memorizzati nell'unità a disco del computer. Successivamente, il modulo completo potrà essere caricato nella memoria del sistema. Quindi il procedimento per lo sviluppo e il debugging del programma è essenzialmente identico a quello descritto in precedenza nel par. 1.2 per lo sviluppo del cross-software. La differenza è che il software di sviluppo *residente* associato con la programmazione in linguaggio assembler è eseguibile direttamente sul sistema basato sull'MC68020 che sarà impiegato per l'esecuzione del programma. Per utilizzare il software residente, il computer basato sull'MC68020 deve comprendere un'unità di memorizzazione su disco con relativo hardware e software. Il sistema operativo viene usato per caricare il programma nella memoria dall'unità a disco e per eseguirlo. La sequenza di comandi per svolgere questo compito dipende interamente dalle caratteristiche del sistema operativo impiegato.

I sistemi operativi della Motorola, quale il VERSAdos, comprendono programmi di servizio (*utilities*) per consentire all'operatore di effettuare il debugging di programmi applicativi. Uno di tali programmi è SYMbug (*symbolic debugger*: debugger simbolico). Esso dispone di funzioni o comandi per caricare un programma dall'unità a disco e per eseguirlo ai fini del debugging. I comandi per il debugging sono simili a quelli definiti in precedenza per il monitor 133BUG.

ESERCIZI

5.1.1 Le caratteristiche di esecuzione passo-passo e di breakpoint di un debugger sono simili. Si discuta l'impiego di ciascuna tecnica e le si confrontino.

5.1.2 Un certo numero di sistemi di sviluppo è disponibile per l'MC68020. Si scelgano alcuni sistemi e li si confrontino per quanto concerne le rispettive capacità di sviluppo e debugging sia dell'hardware che del software. Si consiglia di consultare la letteratura tecnica delle case produttrici.

5.1.3 Si consideri l'impiego e i vantaggi e gli svantaggi di un sistema di sviluppo del software avente le seguenti caratteristiche:
(a) Un programma monitor con un assemblatore di una sola riga e un disassemblatore;
(b) Un cross-assemblatore ed un simulatore eseguiti su un grande computer;
(c) Un sistema basato su disco con un assemblatore residente;
(d) Un assemblatore con un caricatore di collegamento, rispetto ad un caricatore assoluto. Quest'ultimo non è in grado di rilocare i programmi.

5.1.4 Si confronti il procedimento di test e debugging che impiega un linguaggio ad alto livello con i metodi disponibili al programmatore in linguaggio assembler. Come esempio specifico, si faccia uso del semplice programma mostrato precedentemente in questo paragrafo.

5.2 CARATTERISTICHE DEL LINGUAGGIO ASSEMBLER

Le istruzioni di programma-sorgente della memoria, così come sono elaborate dall'assemblatore, consistono di stringhe di caratteri ASCII combinati per formare simboli. Questi simboli sono costruiti in base alle regole del linguaggio. Ogni istruzione consiste di quattro *campi:* un'etichetta, un'operazione, uno o più operandi, e un commento. I campi sono separati da spazi o da altri delimitatori a seconda del *formato* richiesto. Per esempio, l'istruzione:

```
INIT    MOVE.L    #0,D1    ;AZZERA SOMMA
```

consiste di: un'etichetta INIT; un'istruzione mnemonica MOVE, che rappresenta l'operazione; un campo di operazione (#0,D1); ed un commento. In questo esempio, il delimitatore tra i campi è il carattere di spazio o "blank": almeno uno spazio è richiesto per separare i campi; comunque, possono essere usati più spazi, come in questo caso. Tale formato è noto come *formato di campo libero.*

Normalmente, l'assemblatore scandisce dapprima ciascuna istruzione del programma-sorgente per verificare la validità del formato e dei simboli impiegati. Viene riportata una segnalazione di errore se il formato non è corretto o se l'operazione non è un'istruzione del processore né una direttiva dell'assemblatore.

In questo paragrafo, la discussione del linguaggio assembler dell'MC68020 è suddivisa in tre parti. La prima tratta la costruzione delle istruzioni-sorgente che

rappresentano le istruzioni eseguibili per il processore. La seconda parte tratta le direttive dell'assemblatore, che controllano la modalità operativa dell'assemblatore stesso. Le caratteristiche discusse in questo paragrafo sono necessarie per la creazione di programmi utili, sebbene la maggior parte degli assemblatori abbiano molte capacità addizionali, che saranno descritte nel sottopar. 5.2.3.

5.2.1 Formati delle istruzioni

Le istruzioni-sorgente elaborate dall'assemblatore devono attenersi ad un ben preciso formato, che definisce l'ordine e la relazione degli elementi nell'istruzione. Gli assemblatori dell'MC68020 riconoscono istruzioni-sorgente composte dai seguenti campi:

(a) Campo di etichetta

(b) Campo di codice operativo o di direttiva

(c) Campo di operando/i

(d) Campo di commento

Ogni istruzione del linguaggio assembler consiste di questi elementi, separati da spazi.

La Tab. 5.2 illustra il formato di una generica istruzione del linguaggio assembler, in cui i campi opzionali sono racchiusi tra parentesi quadre. Se un simbolo di asterisco (*) viene incontrato nella prima colonna di un'istruzione, questa viene considerata interamente come un commento. Se un carattere diverso compare nella prima colonna, allora il simbolo è considerato l'inizio di un'etichetta, che comprende da uno a otto caratteri alfanumerici. Nella maggior parte degli assemblatori, il primo carattere di un simbolo dev'essere necessariamente una lettera (A-Z), anche se per altri assemblatori valgono convenzioni diverse. Se non c'è un'etichetta, la prima colonna deve contenere uno spazio bianco ("blank") se sono presenti altri campi.

Tab. 5.2 *Formato del linguaggio assembler.*

Campo di etichetta	**Campo di codice operativo e di direttiva**	**Campo di operando/i**	**Campo di commento**
[<ETICHETTA>]	<cod.op.> o <direttiva>	[<operando1>,[<operando2>]]	[<commento>]

Note:
1. Un asterisco nella colonna 1 indica una riga di commento.
2. Le parentesi angolari indicano qualsiasi simbolo valido.
3. Le parentesi quadre indicano un campo opzionale.

Il successivo campo incontrato viene interpretato come il codice mnemonico di un'istruzione o una direttiva dell'assemblatore. Per esempio, in un'istruzione senza etichetta, come:

```
MOVE.W    D1,D2    ;COMMENTO
```

il codice mnemonico dell'istruzione deve partire dalla colonna 2 o oltre. Almeno uno spazio deve precedere gli operandi ed il commento. Il punto e virgola non è necessario prima del commento, ma è usato qui per migliorare la leggibilità.

Etichette. L'etichetta è facoltativa per la maggior parte delle istruzioni e direttive. Quando viene usata, essa rappresenta un indirizzo. All'etichetta viene assegnato il valore del contatore di locazione della riga in cui appare. In istruzioni come:

```
QUI    MOVE.W    D1,D3
```

l'etichetta QUI definisce la locazione dell'istruzione nella memoria dopo che il programma è stato caricato e può essere usata per definire l'inizio di un segmento di programma per riferimenti successivi.

Esempio 5-3

La Fig. 5.6 mostra alcuni esempi di etichette usate come indirizzi. Come discusso nel precedente par. 5.1, il programma somma quattro valori ed ottiene un risultato. Il programma parte dalla locazione $10000 ed inizializza la somma a zero quando viene eseguito. Il simbolo INIT è un'etichetta applicata alla prima istruzione ed ha il valore $10000.

```
                             1.         TTL      FIGURA 5.6
                             2.         LLEN     100             ;LUNGHEZZA DELLA RIGA
00010000                     3.         ORG      $10000          ;ORIGINE NELLA MEMORIA
                             4.  *
                             5.  * SOMMA 4 NUMERI DI 16 BIT MEMORIZZATI NELLE LOCAZIONI
                             6.  * DA $20000 A £20006. RIPORTA LA SOMMA IN D1[15:0].
                             7.  *
00010000  7200               8.  INIT   MOVE.L   #0,D1           ;SOMMA ZERO
00010002  227C 00020000      9.         MOVEA.L  #$20000,A1      ;INDIRIZZO DEL PRIMO NUMERO
00010008  7404              10.         MOVE.L   #4,D2           ;PONE IL CONTATORE A 4
                            11.  *
                            12.  * DEFINISCE IL CICLO PER LA SOMMA DEI VALORI
                            13.  *
0001000A  D259              14.  LOOP   ADD.W    (A1)+,D1        ;SOMMA I NUMERI
0001000C  5342              15.         SUB.W    #1,D2           ;DECREMENTA IL CONTATORE
0001000E  66 FA             16.         BNE      LOOP            ;FINCHE' (D2)=0
                            17.  *
00010010  4E4F              18.         TRAP     #15             ;RITORNA AL MONITOR
00010012  0063              19.         DC.W     $0063
00010014                    20.         END
```

Fig. 5.6 *Uso di etichette.*

Un altro programma (non mostrato) potrebbe usare l'istruzione

```
JMP        INIT
```

per avviare l'esecuzione di questo segmento. L'etichetta LOOP (ciclo) contrassegna la prima istruzione di una sequenza ripetuta. Il ciclo sarà eseguito quattro volte, finché il contatore (D2) non avrà raggiunto lo zero in seguito ai successivi decrementi. Questa etichetta si riferisce semplicemente alla sequenza di addizione e non dovrebbe avere altri riferimenti oltre a quello dell'istruzione BNE all'interno del ciclo.

Codici operativi. Il secondo campo dell'istruzione-sorgente deve contenere un codice mnemonico d'istruzione o una direttiva di assemblatore. Quando gli operandi richiedono che sia specificata una lunghezza, una specificazione di lunghezza è inclusa come parte del campo d'istruzione. Tale specificazione è preceduta da un punto (.) che viene scritto dopo il codice operativo, e consiste della lettera B, W o L per specificare che si tratta di un byte, o di una word, o di una longword, rispettivamente. Per esempio, l'istruzione

```
MOVE.W     D1,D2
```

definisce operandi con lunghezza di word.

Operandi. L'accesso alle locazioni degli operandi per le istruzioni avviene secondo la modalità d'indirizzamento per ciascun operando. I formati generali degli operandi sono mostrati nella Tab. 5.3. Alcune istruzioni non richiedono alcun operando. Altre si riferiscono implicitamente ai registri del processore, e la loro esecuzione può causare la modifica del contatore di programma, del puntatore di stack o del registro di stato. Le istruzioni TRAP e STOP richiedono un valore immediato nella forma di un numero decimale o esadecimale.

La maggior parte delle istruzioni richiedono che gli operandi siano specificati dalle modalità d'indirizzamento. Le istruzioni a singolo indirizzo contengono la specificazione di un solo operando. Le istruzioni a doppio indirizzo contengono due operandi, che sono separati da un virgola nel campo di operando. I registri d'indirizzo o di dati del processore sono designati simbolicamente dalla lettera A o D, seguita dal numero del registro. Pertanto l'istruzione

```
MOVE.W     A1,D1
```

designa A1 come registro di sorgente e D1 come destinazione. L'indirizzamento indiretto è specificato racchiudendo tra parentesi il simbolo del registro d'indirizzo, come nell'istruzione

```
MOVE.W     (A1),D2
```

che produce in D2 la copia della locazione puntata da (A1). Le modalità d'indirizzamento saranno descritte in dettaglio nel par. 5.3.

Tab. 5.3 *Riferimenti di istruzioni dell'MC68020.*

Operando	Formato	Riferimento o operando tipico	Esempio
Nessuno	OPR	Dispositivo esterno	RESET
Implicito	OPR	PC, SP o SR	NOP, TRAPV, RTS
Immediato	OPR <valore>	Controllo del processore o istruzioni che richiedono un valore	TRAP, STOP
Singolo	OPR <indirizzo>	Indirizzo relativo Indirizzo di istruzione Indirizzo di operando	BRA JMP CLR, NEG
Doppio	OPR <valore>,<destinazione> OPR <sorgente>,<destinazione>	Valore immediato alla destinazione o doppio indirizzo	ADD, MOVE

Note:
1. OPR è qualsiasi codice operativo valido.
2. Sono possibili alcune lievi varianti del formato mostrato.

Esempio 5-4

La Fig. 5.7 è un listato di assemblatore, che mostra un certo numero di istruzioni per illustrare la specificazione di operandi. Le istruzioni RESET e RTS (*ReTurn form Subroutine:* ritorna da subroutine) non richiedono operandi. Un'istruzione TRAP deve avere il numero di trappola specificato come valore immediato. Tali istruzioni hanno requisiti unici per la specificazione dell'operando.

Le istruzioni a singolo indirizzo, come CLR, richiedono un indirizzo nel campo di operando. L'indirizzo può essere specificato da qualsiasi modalità d'indirizzamento valida per la particolare istruzione. Sono mostrati vari esempi per specificare l'operando per l'istruzione CLR.

L'istruzione MOVE a doppio indirizzo è mostrata con le varie modalità d'indirizzamento usate per specificare l'operando di sorgente. La destinazione è un registro in ogni esempio, sebbene l'istruzione MOVE non ammetta altre modalità d'indirizzamento per l'operando di destinazione.

```
                          1.         TTL      FIGURA 5.7
                          2.         LLEN     100
00010000                  3.         ORG      $10000
                          4.   *
                          5.   * ISTRUZIONI MISCELLANEE
                          6.   *
00010000  4E70            7.         RESET
                          8.   *
00010002  4E4F            9.         TRAP     #15
                         10.   *
00010004  4E75           11.         RTS
                         12.   *
                         13.   * SINGOLO INDIRIZZO
                         14.   *
00010006  4241           15.         CLR.W    D1                ;DIRETTO REG. DATI
00010008  4278 1000      16.         CLR.W    $1000             ;ASSOLUTO
0001000C  4251           17.         CLR.W    (A1)              ;INDIRETTO
0001000E  4261           18.         CLR.W    -(A1)             ;PREDECREMENTO
00010010  4259           19.         CLR.W    (A1)+             ;POSTINCREMENTO
00010012  4269 0002      20.         CLR.W    2(A1)             ;INDIRETTO CON SPOST.
00010016  4271 1002      21.         CLR.W    (2,A1,D1.W)       ;INDIRETTO CON INDICE
0001001A  4271 A802      22.         CLR.W    (2,A1,A2.L)
                         23.   *
                         24.   * DOPPIO INDIRIZZO (INDIRIZZO DI SORGENTE SPECIFICATO)
                         25.   *
0001001E  3401           26.         MOVE.W   D1,D2             ;DIRETTO REG. DATI
00010020  3409           27.         MOVE.W   A1,D2             ;DIRETTO REG. INDIRIZZO
00010022  3428 1000      28.         MOVE.W   $1000,D2          ;ASSOLUTO
00010026  3411           29.         MOVE.W   (A1),D2           ;INDIRETTO
00010028  3421           30.         MOVE.W   -(A1),D2          ;PREDECREMENTO
0001002A  3419           31.         MOVE.W   (A1)+,D2          ;POSTINCREMENTO
0001002C  3429 0002      32.         MOVE.W   2(A1),D2          ;INDIRETTO CON SPOST.
00010030  3431 1002      33.         MOVE.W   (2,A1,D1.W),D2    ;INDIRETTO CON INDICE
00010034  343C 0005      34.         MOVE.W   #5,D2             ;IMMEDIATO
00010038  3439 00010040  35.         MOVE.W   *+8,D2            ;RELATIVO
                         36.   *
0001003E  4E4F           37.         TRAP     #15               ;RITORNO
00010040  0063           38.         DC.W     $0063
00010042                 39.         END
```

Fig. 5.7 *Esempi di specificazioni di operando.*

Espressioni come operandi. L'assemblatore riconosce certi simboli nel campo di operando. Un simbolo può designare un indirizzo assoluto, un valore immediato, o qualsiasi altro operando valido. Un'*espressione* è una combinazione di simboli, costanti, operatori algebrici e parentesi che l'assemblatore valuta per determinare l'indirizzo o il valore dell'operando.

Per specificare un valore costante, talvolta chiamato *letterale*, s'impiega il modo d'indirizzamento immediato. Per la maggioranza degli assemblatori, le costanti possono rappresentare sia numeri che caratteri ASCII. Una costante è la più semplice forma di espressione ed è specificata usando le definizioni fornite nella Tab. 5.4. Una costante numerica può essere qualsiasi valore decimale o esadecimale che può essere rappresentato come un intero di 8 bit, 16 bit o 32 bit. La specificazione della dimensione dell'istruzione determina la lunghezza appropriata. Un numero decimale è definito come una stringa di cifre decimali, mentre un numero esadecimale è definito da un segno di dollaro ($) seguito da una stringa di cifre esadecimali. Quindi l'istruzione

```
MOVE.W    #$2000,D1
```

definisce il valore esadecimale 2000 di 16 bit come l'operando immediato di sorgente.

Tab. 5.4 *Simboli dell'assemblatore per le espressioni.*

Formato simbolico	**Interpretazione**
$<Numero>	Numero esadecimale
<Numero>	Numero decimale
'<Stringa>'	Stringa di caratteri ASCII
#<Numero>	Operando immediato
#'<Stringa>'	Operando immediato
Nelle espressioni:	
+	Addizione
–	Sottrazione
*	Moltiplicazione
/	Divisione
( )	Raggruppamento

Un letterale ASCII consiste di un gruppo di caratteri ASCII — al massimo quattro — racchiuso da apostrofi. Per esempio, la stringa 'ABCD' è riconosciuta dall'assemblatore e convertita nel codice ASCII

41 42 43 44

che occupa quattro byte quando viene memorizzata. Una stringa di caratteri più lunga non può essere usata come un valore letterale in un'espressione, poiché la dimensione è limitata dai registri di 32 bit dell'MC68020.

In un'espressione si possono impiegare gli operatori per l'addizione, la sottrazione, la moltiplicazione e la divisione. Il risultato di qualsiasi operazione aritmetica è un valore intero di 32 bit. L'uso del meno unario (–) è riconosciuto come un mezzo per definire numeri negativi. Per esempio, il valore immediato –1 è calcolato nell'istruzione

```
MOVE.W    #-1,D1
```

ed è memorizzato come $FFFF con l'istruzione. Una specificazione equivalente è:

```
MOVE.W    #$FFFF,D1
```

dove il valore $FFFF è il numero in complemento a 2 in 16 bit.

Esempio 5-5

Il breve segmento di programma in Fig. 5.8 illustra vari usi di etichette ed espressioni. Le etichette START e ENDLP servono a definire i valori dell'inizio e della fine di un gruppo di espressioni nell'esempio. Le istruzioni MOVEA e MOVE inizializzano l'indirizzo e il contatore, rispettivamente, quando il programma viene eseguito. Quando l'addizione sarà stata completata, il risultato sarà in (D1). La lunghezza del segmento viene calcolata come 14 byte (sette word) in (D3). La lunghezza in word in (D4) è metà della lunghezza in byte. Se le istruzioni di linguaggio-macchina fossero trasferite nella memoria senza riassemblaggio, allora le espressioni che fanno riferimento a START come un indirizzo o un operando di sorgente risulterebbero errate. Tuttavia, i calcoli delle lunghezze sarebbero corretti.

La stessa istruzione alla locazione START viene trasferita in D5 quando l'operando di sorgente in un'istruzione MOVE specifica soltanto l'etichetta. La forma immediata #START seleziona l'indirizzo.

```
                            1.         TTL      FIGURA 5.8
                            2.         LLEN     100
00010000                    3.         ORG      $10000
                            4.  *
                            5.  * USO DI ETICHETTE E DI ESPRESSIONI
                            6.  *
                            7.
00010000  272C 00020000     8.  START  MOVEA.L #$20000,A1            ;PRIMO INDIRIZZO
00010006  7404              9.         MOVE.L  #4,D2                 ;CONTATORE
                           10.  *
                           11.  *      LA SOMMA VIENE AGGIUNTA AL VALORE INIZIALE IN D1
                           12.  *
00010008  D259             13.  LOOP   ADD.W   (A1)+,D1              ;SOMMA QUATTRO NUMERI
0001000A  5382             14.         SUBQ.L  #1,D2
0001000C  66 FA            15.         BNE     LOOP
0001000E  4E71             16.  ENDLP  NOP
                           17.  *
                           18.  *  LUNGHEZZA DEL PROGRAMMA IN BYTE
                           19.  *
00010010  760E             20.         MOVE.L  #(ENDLP-START),D3     ;D3 = 14
                           21.  *
                           22.  *  LUNGHEZZA DEL PROGRAMMA IN WORD
                           23.  *
00010012  7807             24.         MOVE.L  #(ENDLP-START)/2,D4   ;(D4) = 7
                           25.  *
                           26.  *  CONTENUTO DI START (L'ISTRUZIONE) IN D5
                           27.  *
00010014  2A39 00010000  · 28.         MOVE.L  START,D5              ;(D5) = 2227C 0002
                           29.  *
                           30.  *  INDIRIZZO DI START IN A2
                           31.  *
0001001A  247C 00010000    32.         MOVEA.L #START,A2             ;(A2) = 10000
                           33.  *
                           34.  *  STRINGA ASCII IN D6
                           35.  *
00010020  232C 444F4E45    36.         MOVE.L  #'FINE',D6            ;(D6) = 444F 4E45
                           37.  *
00010026  4E4F             38.         TRAP    #15
00010028  0063             39.         DC.W    $0063
0001002A                   40.         END
```

Fig. 5.8 *Uso di etichette e di espressioni.*

L'istruzione MOVE finale trasferisce la stringa ASCII 'FINE' in D3. Si noti che dev'essere indicato il modo immediato, altrimenti il valore sarebbe interpretato erroneamente come un indirizzo. Si potrebbe utilizzare un'opportuna routine di I/O per far stampare la stringa, al fine d'indicare il completamento del programma.

5.2.2 Direttive dell'assemblatore

I simboli mnemonici per i codici operativi delle istruzioni e quelli per le varie modalità d'indirizzamento fanno parte della *tabella dei simboli* interna usata dall'assemblatore per convertire in linguaggio-macchina le istruzioni-sorgente. I simboli definiti dall'utente, come le etichette, sono usati per far riferimento a istruzioni o dati entro il programma in linguaggio assembler. L'assemblatore tiene nota automaticamente delle locazioni e degli "offset" associati col programma in linguaggio-macchina.

L'impiego di forme simboliche come indirizzi di istruzioni o come operandi risulta prezioso nella scrittura di programmi in linguaggio assembler ed è uno dei principali vantaggi di questo linguaggio rispetto al linguaggio-macchina. Comunque, la maggior parte degli assemblatori assistono il programmatore in altri modi, disponendo di *direttive di assemblatore*, che sono in effetti delle istruzioni rivolte all'assemblatore anziché al processore. L'azione causata da ciascuna direttiva viene svolta soltanto in fase di assemblaggio del programma-sorgente. Le principali categorie di direttive riguardano il controllo dell'assemblaggio, la definizione dei simboli, la definizione dei dati e l'allocazione della memoria, e il controllo del listato, come mostrato nella Tab. 5.5 a pagina seguente.

Controllo dell'assemblaggio. Il contatore di locazione dell'assemblatore inizia normalmente col valore $0000 per indicare la locazione della prima istruzione eseguibile. Questo contatore viene incrementato della quantità appropriata quando un'istruzione viene assemblata. Se il programma in linguaggio-macchina venisse caricato nella memoria alla locazione $0000 ed eseguito, il contatore di programma seguirebbe la medesima sequenza del contatore di locazione mentre ciascuna istruzione viene progressivamente eseguita.

Il caricamento di un programma a partire dalla locazione $0000 non è consigliabile nei sistemi dell'MC68020, poiché le aree più basse della memoria sono riservate ai vettori dell'MC68020. L'impiego della direttiva ORG (origine) consente al programmatore di definire il valore iniziale del contatore di locazione e, di conseguenza, il primo indirizzo del programma nella memoria. Negli esempi precedenti di questo capitolo, si è è usata la direttiva ORG per indicare che la locazione esadecimale $10000 doveva servire a memorizzare la prima istruzione in linguaggio-macchina di un programma.

Tab. 5.5 *Direttive dell' assemblatore.*

DIRETTIVA E FORMATO	SIGNIFICATO
Controllo dell'assemblaggio	
ORG <espressione>	Origine
END	Fine sorgente
Definizione dei simboli	
<etichetta> EQU <espressione>	Uguaglia valore di <etichetta>
Definizione dei dati e memorizzazione	
[<etichetta>] DC.<1> <valore/i>	Definisce costante/i
[<etichetta>] DS.<1> <numero>	Riserva memoria
Controllo del listato	
LLEN <N>	Lunghezza della riga
LIST	Produce il listato (default)
NOLIST	Nessun listato
SPC <N>	<N> righe bianche
PAGE	Salto-pagina

Note:
1. Le parentesi quadre indicano un campo opzionale.
2. <1> denota B, W o L.

Il formato della direttiva ORG è il seguente:

ORG <espressione>

in cui <espressione> ha il medesimo significato definito in precedenza. Quando l'assemblatore incontra questa direttiva, il contatore di locazione viene "caricato" col relativo valore, in maniera simile alla modifica del contenuto del PC prodotta da un'istruzione di salto (JMP). La direttiva ORG può comparire dovunque nel programma-sorgente e può essere usata, ad esempio, per suddividere il programma in sezioni di istruzioni e di dati. Ciò è particolarmente utile se le istruzioni devono essere contenute in una memoria a sola lettura, mentre i dati sono contenuti in una memoria che consente la scrittura, a partire da un'altra locazione iniziale. In alternativa, si potrebbe specificare l'indirizzo di partenza per l'editor di collegamento (*linkage editor*), se l'assemblatore produce indirizzi rilocabili.

Un'altra direttiva di controllo dell'assemblaggio è END (fine), che dev'essere sempre l'ultima istruzione-sorgente in un programma. Essa fa sì che l'assemblato-

re arresti la sua scansione del programma dall'alto verso il basso. Qualunque istruzione-sorgente posta dopo la direttiva END non sarà elaborata dall'assemblatore.

Definizione dei simboli. Una direttiva EQU serve per uguagliare un numero ad un simbolo. Il valore può rappresentare un indirizzo o una costante. Il suo formato è:

```
<etichetta>          EQU     <espressione>
```

dove a <etichetta> viene assegnato il valore di <espressione> quando questa riga viene assemblata. L'espressione può contenere un'etichetta, purché questa sia stata definita precedentemente nel programma. Quindi la seguente riga di programma:

```
TTYOUT               EQU     $7FFF
```

assegna il valore $7FFF al simbolo TTYOUT. Lo scopo potrebbe essere quello di definire l'indirizzo di un buffer di uscita usando un codice mnemonico. Quindi una riga come la seguente:

```
MOVEA.L   #TTYOUT,A1
```

trasferisce il numero $7FFF al registro d'indirizzo A1. Se per la sorgente non s'impiega il modo immediato, allora il contenuto della locazione viene trasferito. L'istruzione

```
MOVE.B    D1,TTYOUT
```

trasferirebbe un byte da D1 alla locazione $7FFF, che potrebbe essere, ad esempio, la locazione di un buffer di uscita.

Un importante vantaggio della direttiva EQU appare evidente quando viene definito un valore a cui si fa riferimento più volte all'interno di un programma. Se l'indirizzo TTYOUT dovesse essere modificato in un successivo assemblaggio, il riassemblaggio con una nuova EQU posta all'indirizzo corretto modificherebbe il valore in questione dappertutto nel programma. Questa condizione potrebbe verificarsi se il programma è eseguito su diversi sistemi, ciascuno con differenti locazioni di buffer.

La direttiva EQU può servire anche per assegnare nomi utili alle costanti matematiche, come in:

```
UNK        EQU     1024
```

che definisce "UNK" come valore costante 1024 decimale. Come altro esempio, l'etichetta MAXMEM potrebbe essere uguagliata al massimo spazio di memoria disponibile per un sistema. Tale valore potrebbe essere modificato, se occorre,

allorché il programma dev'essere riassemblato su un nuovo sistema. L'unico inconveniente è che il valore definito dalla direttiva EQU è noto solamente all'assemblatore e non esiste nella memoria; pertanto, esso non può essere modificato se non si procede al riassemblaggio.

Direttive di dati. Le due direttive DC (*Define Constant:* definisci costante) e DS (*Define Storage:* definisci memoria) sono disponibili per inizializzare i valori nella memoria e per riservare spazio nella memoria. La direttiva DC è simile alla dichiarazione DATA in FORTRAN, in cui alle variabili definite sono assegnati dei valori iniziali. La direttiva DS è simile alla dichiarazione DIMENSION, che riserva lo spazio per le variabili, a cui non assegna però alcun valore.

La direttiva DC fa sì che l'assemblatore memorizzi valori specificati nella locazione o nelle locazioni associate col valore del contatore di locazione nell'istante in cui la direttiva DC viene incontrata durante l'assemblaggio. Quando il programma in linguaggio-macchina viene caricato nella memoria, le locazioni interessate hanno il valore iniziale specificato. Per esempio, l'istruzione

```
INITV    DC.W    20
```

fa sì che il valore decimale 20 occupi una word nella locazione INITV. Tuttavia, se il programma viene eseguito più volte ed il valore di INITV cambia tra le esecuzioni, qualsiasi istruzione del programma che dipende dal valore iniziale 20 potrebbe non fornire i risultati corretti in fase di esecuzione.[4] Quindi la direttiva DC non dovrebbe essere mai usata per inizializzare un valore che può essere modificato dopo che il programma è stato caricato nella memoria se le esecuzioni di più programmi dipendono dai valori iniziali. Un metodo migliore per l'inizializzazione dei valori consiste nel riservare spazio ai valori con una direttiva DS e poi assegnare i valori iniziali con istruzioni eseguibili.

Entrambe le direttive DC e DS richiedono una specificazione di lunghezza (B, W o L). La specificazione della lunghezza determina se sono riservati byte, word o longword. Quindi la direttiva

```
         DS.W    $10
```

riserva 16 word nella memoria. La lunghezza di ogni costante definita dalla direttiva DC è determinata dalla specificazione della dimensione. Per esempio, la direttiva

```
INDIR1   DC.W    ETICH+1
```

memorizzerà l'indirizzo di ETICH più 1 nella locazione di word all'indirizzo INDIR1.

---

[4] Il contenuto della locazione potrebbe variare se un altro programma scrive un valore diverso nella locazione. Ciò può avvenire in sistemi in cui la medesima area di memoria è condivisa da programmi eseguiti in tempi diversi. Un sistema in time-sharing è un esempio di circostanza in cui un'area di memoria potrebbe essere condivisa dagli utenti.

Controllo del listato. L'ultimo gruppo di direttive nella Tab. 5.5 indica alcune opzioni disponibili per definire il formato del listato prodotto dall'assemblatore. La direttiva LLEN (lunghezza della riga) determina il numero di caratteri nelle righe di stampa. LLEN 72 è tipicamente usata per unità CRT, ma righe più lunghe, con più di 72 caratteri, possono essere usate per le stampanti di riga. La direttiva SPC fa apparire il numero specificato di righe bianche sulla stampa per migliorare la leggibilità. Di solito sono disponibili altre direttive, come PAGE, per definire il formato del listato. La direttiva PAGE produce un avanzamento all'inizio della pagina successiva (salto-pagina) ogni volta che viene incontrata. La lunghezza della pagina dipende dalla stampante utilizzata ed è generalmente definita come parametro nel sistema operativo.

Generalmente, ogni opzione per il listato ha un valore prefissato di cui si può specificare l'opposto. Per esempio, la direttiva NOLIST nella Tab. 5.5 causa l'omissione delle istruzioni successive dal listato. Il suo opposto, la direttiva LIST, è il valore prefissato e non dev'essere specificato, a meno che l'opzione NOLIST non debba essere rovesciata. Quindi la sequenza:

```
LIST
(segmento I)
NOLIST
(segmento II)
LIST
(segmento III)
END
```

fa apparire nel listato i segmenti I e III di un programma, ma non il segmento II.

Esempio 5-6

La Fig. 5.9 illustra un programma che impiega un certo numero di direttive di assemblatore. Le prime tre direttive creano un titolo, definiscono la lunghezza della riga e fissano l'origine a $10000, rispettivamente. Le direttive EQU definiscono la costante ONE, l'indirizzo iniziale del programma per la seconda direttiva ORG ed il valore costante ($0063) per l'istruzione TRAP #15.

La prima direttiva ORG definisce l'area per la memorizzazione dei dati, a partire dalla locazione $10000. Il programma eseguibile inizia dalla locazione $20000, come specificato dalla direttiva

```
ORG     PROGRAM
```

Il programma azzera le locazioni tra COMMON e l'ultima locazione impiegata per i dati.

```
---------------------------------------------------------------------------

                          1.          TTL     FIGURA 5.9
                          2.          LLEN    100
00010000                  3.          ORG     $10000
    # 00000001            4.  ONE     EQU     1                  ;UNA COSTANTE
    # 00020000            5.  PROGRAM EQU     $20000             ;INDIRIZZO INIZ.
    # 00000063            6.  RETURN  EQU     $0063
                          7.  *
                          8.  *       AREA DATI
                          9.  *
00010000  0A 05 07       10.  INITDT  DC.B    10,5,7             ;BYTE - DECIMALI
          -- boundary align
00010004  0000000A       11.          DC.L    10,5,7             ;LONGWORD
          00000005
          00000007
00010010  FF 10 AF       12.          DC.B    $FF,$10,$AF        ;BYTE HEX
          -- boundary align
00010014  0000000F       13.          DC.L    $FF,$20,$AE        ;LONGWORD
          00000020
          000000AE
00010020  41 42 43 44 45 14.          DC.B    'ABCDEFGH'         ;BYTE ASCII
          46 47 48
00010028  41 00 00 00 42 15.          DC.L    'A','BC'           ;LONGWORD
          43 00 00
                         16.  *
00010030  00010000       17.  INITADD DC.L    INITDT             ;INPUT INDIRIZZO
00010034  <14>           18.  COMMON  DS.W    10                 ;10 WORD
00010048  <20>           19.  HEXVAL  DS.W    $10                ;16 WORD
00010068  <3>            20.  BYTES   DS.B    3                  ;3 BYTE
          -- boundary align
0001006C  <0>            21.          DS      0                  ;CONFINE PARI
                         22.  *
                         23.  *  LA LUNGHEZZA E' CALCOLATA COME L'ULTIMA LOCAZIONE DI BYTE
                         24.  *  MENO LA PRIMA LOCAZIONE DI DATI:  (BYTES+3)-COMMON
                         25.  *
    # 00000037           26.  LENGTH  EQU     (BYTES+3-COMMON)  ;NUMERO DI BYTE
                         27.  *
00020000                 28.          ORG     PROGRAM
                         29.  *
                         30.  *       CANCELLA I VALORI COMUNI
                         31.  *
00020000  123C 0037      32.  BEGIN   MOVE.B  #LENGTH,D1         ;CONTATORE
00020004  227C 00010034  33.          MOVEA.L #COMMON,A1         ;INDIRIZZO DELLA PRIMA WORD
                         34.  *
0002000A  12FC 0000      35.  LOOP    MOVE.B  #0,(A1)+           ;CANCELLA L'AREA COMUNE
0002000E  5301           36.          SUBQ.B  #ONE,D1
00020010  66 F8          37.          BNE     LOOP
                         38.  *
00020012  4E4F           39.          TRAP    #15
00020014  0063           40.          DC.W    RETURN
00020016                 41.          END

---------------------------------------------------------------------------
```

Fig. 5.9 *Uso delle direttive di assemblatore.*

Le direttive DC definiscono un certo numero di costanti nell'area di dati. Si noti che la riga

INITADD DC.L INITDT

inizializza INITADD con l'indirizzo di INITDT. L'istruzione

MOVE.L #INITDT,INITADD

svolgerebbe la medesima funzione, ma soltanto durante l'esecuzione del programma.

In totale sono riservati 55 byte dalle varie direttive DS; si noti che i medesimi risultati si sarebbero ottenuti con la dichiarazione:

```
DS.B 55
```

Tuttavia, il metodo adottato qui è preferibile poiché si presume che in qualche altro segmento di programma (non mostrato) si faccia riferimento ai singoli blocchi di variabili (COMMON, HEXVAL, BYTES). La direttiva finale:

```
DS  0
```

serve ad allineare su un confine di word (pari) l'indirizzo della locazione successiva. Ciò viene effettuato automaticamente dall'assemblatore utilizzato per l'esempio.

5.2.3 Caratteristiche avanzate degli assemblatori

La maggior parte dei moderni assemblatori, compreso il Resident Assembler della Motorola e il cross-assemblatore della Quelo, dispongono di un certo numero di caratteristiche per migliorare la struttura del programma e la leggibilità. Tali caratteristiche sono state perlopiù sempre presenti negli assemblatori per grandi computer, ma solo di recente sono divenute disponibili per molti sistemi basati su microcomputer. Qui saranno discusse solo a livello introduttivo, per portare a conoscenza del lettore queste utili tecniche. Si consiglia a chiunque voglia intraprendere un progetto di sviluppo abbastanza complesso, che richieda il linguaggio assembler, di esplorare queste capacità per il particolare computer che s'intende utilizzare. La maggior parte delle caratteristiche viene attivata includendo nel programma le opportune direttive di assemblatore. In qualche caso, è necessario l'editor di collegamento per completare le operazioni.

La Tab. 5.6 elenca un certo numero di capacità di determinati assemblatori che vanno oltre la conversione di ciascuna istruzione in linguaggio assembler in un'istruzione di linguaggio-macchina. Esse sono elencate in ordine alfabetico nella tabella, ma per il resto non hanno alcuna priorità d'importanza. Il loro impiego dipende dall'applicazione.

Tab. 5.6 *Caratteristiche avanzate degli assemblatori.*

Tipo	Scopo o definizione
Assemblaggio condizionato	Consentire l'assemblaggio condizionato
Riferimenti esterni	Riferimenti a variabili o a programmi in un altro modulo-oggetto
Libreria	Una raccolta di moduli-oggetto considerati come un'unità
Macroistruzioni	Un gruppo di istruzioni in linguaggio-assembler riferite tramite il nome

La tecnica di *assemblaggio condizionato* consente ad un programmatore di selezionare certe porzioni del programma, che saranno assemblate oppure no a seconda di certe condizioni logiche. Per esempio, la seguente specificazione (direttiva) di assemblaggio condizionato:

```
IFEQ      <espressione>
          <istruzioni del linguaggio-assembler>
ENDC
```

fa sì che le istruzioni tra le direttive IFEQ e ENDC siano assemblate se il valore di <espressione> è uguale (EQ) a zero. Altrimenti, queste istruzioni saranno ignorate dall'assemblatore. Altre condizioni potrebbero includere una specificazione di diversità, maggioranza, o minoranza. Lo scopo è quello di consentire ad un programma di gestire le varianti del medesimo problema di base. La scelta viene effettuata soltanto in fase di assemblaggio del programma. Ogni volta che dev'essere assemblato un diverso segmento di codice, l'<espressione> dev'essere modificata appropriatamente ed il programma dev'essere riassemblato.

La capacità di far riferimento a *variabili esterne* in un programma in linguaggio assembler è d'importanza capitale quando un certo numero di moduli dev'essere collegato per formare un unico modulo di caricamento. Con questa capacità, un programma può far riferimento mediante il nome a programmi o variabili in un altro modulo-oggetto, senza dover specificare l'indirizzo esatto dell'entità riferita. In un modulo, ad esempio, la direttiva

```
XDEF      Y
```

definisce la variabile Y. Un altro modulo includerà l'istruzione

```
XREF      Y
```

per indicare la Y definita altrove. L'editor di collegamento provvederà a gestire tutti i riferimenti esterni e a concatenare i moduli-oggetto separati in un unico modulo-oggetto contiguo e rilocabile.[5] Uno o più di questi moduli possono anche consistere di una "libreria" di moduli.

In questo contesto, una *libreria* (*library:* letteralmente, "biblioteca") è considerata come un gruppo di moduli-oggetto pronti per essere collegati col modulo che fa riferimento a programmi o dati contenuti nella libreria. Ad esempio, in certe applicazioni risulta comodo disporre di una libreria matematica: essa potrebbe consistere di routine per eseguire funzioni quali il calcolo delle radici di un'equazione, e così via. Un programma in linguaggio assembler potrebbe semplicemente far riferimento alle routine tramite il loro nome. Il programma di collegamento effettuerebbe una ricerca nella libreria e vi aggiungerebbe i moduli appropriati per formare il modulo di caricamento completo, prima che il programma venga caricato ed

[5] Il procedimento preciso dipende dal software di sistema impiegato. Esso differisce lievemente tra i vari sistemi di computer.

eseguito. Una libreria viene creata e gestita (modificata dall'aggiunta o dalla rimozione di moduli) da un programma che talvolta viene definito "libraio" (*librarian:* letteralmente, "bibliotecario"). Questo programma opera in collaborazione con i programmi assemblatore ed editor di collegamento, per fornire le informazioni necessarie affinché l'editor di collegamento possa collegare tra loro i moduli appropriati.

Gli assemblatori rendono possibilie una programmazione più modulare e comoda, disponendo di un macroassemblatore per l'assemblaggio delle *macroistruzioni* o semplicemente *macro*. Una macro può essere considerata come il nome di una sequenza di istruzioni in linguaggio assembler. Sotto questo aspetto, una macro è simile ad una subroutine in un linguaggio ad alto livello. Un esempio di struttura di macro potrebbe essere la seguente:

```
MAC1      MACRO
          (istruzioni del linguaggio assembler)
          ENDM
```

dove MAC1 è un'etichetta. Le istruzioni nel corpo della macro sono incluse nel programma in linguaggio assembler ogni volta che viene incontrata l'istruzione MAC1. La macroistruzione ammette anche il passaggio di valori di parametri, in maniera molto simile alla chiamata di una subroutine con vari argomenti.

ESERCIZI

5.2.1 Si scriva una routine che riservi un blocco di memoria di 20 word, a cui dovrà assegnare i valori progressivi da 1 a 20 in fase di esecuzione.

5.2.2 Si trovino gli errori nel seguente programma che deve sommare quattro numeri di 16 bit nelle locazioni da $20000 fino a $20006.

```
ORG       $10000
MOVE.L    $20000,A1
MOVE.L    4,D2
ADD.W     (A1)+,D2
SUB.W     #UNO,D2
BNE       LOOP
UNO       DC.W       1
JMP       RETURN
END
```

5.2.3 Si determinino i valori (se esistono) creati dalle seguenti direttive:

```
(a) DC.B    'N IS'
(b) DC.B    20
(c) QUI     EQU *
(d) DS.L    1
(e) DC.L    ETICH + 2
```

5.2.4 Si supponga che siano state usate delle direttive EQU per assegnare a START il valore $10000 e a END il valore $20000. Si determini il valore calcolato dall'assemblatore per le seguenti espressioni:

(a) START-2
(b) END-START
(c) (END-START)/2
(d) (END-START)/3
(e) 2*END

5.2.5 Si discuta il modo in cui ciascuna delle caratteristiche avanzate dell'assemblatore consente la creazione di programmi per cui risultata facilitata la lettura, il debugging e la modifica (manutenzione).

5.2.6 Si forniscano alcuni esempi che illustrano le circostanze in cui sarebbe utile l'assembler condizionato. Come esempio specifico, si considerino vari modelli del medesimo computer di base, con differenti capacità di memoria ed unità a disco con caratteristiche lievemente diverse. Si supponga che un programma in linguaggio assembler assegni lo spazio di memoria e determini la condizione di riempimento della memoria. Altre routine gestiscono il trasferimento d'ingresso/uscita con l'unità a disco. Si descriva la struttura di tali routine in linguaggio assembler. Come potrebbero essere modificate quando sono pronte per l'esecuzione su differenti modelli del computer?

5.2.7 Si confronti l'impiego di subroutine e di macroistruzioni. Si consideri l'utilizzazione della memoria, come pure la comodità di programmazione e l'efficienza. Una subroutine viene talvolta definita una routine *chiusa*, mentre una macro è denominata routine *aperta*.

5.2.8 Si consideri un grande progetto di programmazione che richieda lo sviluppo e il debugging di un certo numero di moduli in linguaggio assembler, eventualmente scritti da programmatori diversi. Si discutano vari problemi che potrebbero sorgere se il software di sviluppo disponesse soltanto di un assemblatore non rilocante senza caratteristiche avanzate. Per un confronto, in che modo un assemblatore rilocante con caratteristiche avanzate ed un editor di collegamento faciliterebbero lo sviluppo del programma?

5.3 MODALITA' D'INDIRIZZAMENTO PER L'MC68020

Le differenti modalità d'indirizzamento di un processore determinano la varietà dei modi in cui un'istruzione può far riferimento ad un operando o al suo indirizzo. In generale, i processori impiegati in applicazioni avanzate richiedono un gran numero di modalità d'indirizzamento per essere efficaci ed efficienti. L'MC68020 ammette 18 modalità distinte, che lo classificano tra i più potenti microprocessori a questo riguardo. Le modalità d'indirizzamento fondamentali sono state introdotte nel cap. 4, in cui sono stati evidenziati il funzionamento della CPU ed il suo linguaggio-macchina. Nel presente paragrafo, saranno discusse tutte le modalità impiegando la notazione in linguaggio assembler dell'MC68020.

La classificazione generale delle modalità suddette comprende l'indirizzamento diretto di registro, assoluto, indiretto, relativo e immediato. In ciascuna classe, è disponibile un certo numero di varianti per l'MC68020, come indicato nella Tab. 5.7. Per ciascuna modalità, la circuiteria d'indirizzamento del processore calcola l'indirizzo effettivo di qualsiasi operando specificato in un'istruzione allorché tale istruzione viene prelevata e decodificata. La tabella definisce il modo in cui viene calcolato l'indirizzo effettivo (EA) per ciascuna modalità d'indirizzamento.

Tab. 5.7 *Le modalità d'indirizzamento dell'MC68020.*

Modalità	**Calcolo dell'indirizzo effettivo**
INDIRIZZAMENTO DIRETTO DI REGISTRO	
Diretto di registro dati	EA = Dn
Diretto di registro indirizzo	EA = An
INDIRIZZAMENTO DI DATI ASSOLUTO	
Assoluto corto	EA = (word successiva alla word di operazione)
Assoluto lungo	EA = (due word successive alla word di operazione)
INDIRIZZAMENTO INDIRETTO DI REGISTRO	
Indiretto di registro d'indirizzo	EA = (An)
Indiretto con postincremento	EA = (An); (An)←(An)+N
Indiretto con predecremento	(An)←(An)–N; EA = (An)
Indiretto con spostamento	EA = (An)+<d_{16}>
Indiretto con indice (8 bit)	EA = (An)+(Xn)+<d_8>
Indiretto con indice e spostamento di base	EA = (An)+(Xn)+<bd>
INDIRIZZAMENTO INDIRETTO DI MEMORIA	
Indiretto di memoria postindicizzato	EA = [<bd>+An]+(Xn)+<od>
Indiretto di memoria preindicizzato	EA = [<bd>+An+Xn]+<od>
INDIRIZZAMENTO RELATIVO	
Indiretto di PC con spostamento	EA = (PC)+<d_{16}>
Indiretto di PC con indice (8 bit)	EA = (PC)+(Xn)+<d_8>
Indiretto di PC con indice e spostamento di base	EA = (PC)+(Xn)+<bd>
INDIRIZZAMENTO INDIRETTO DI MEMORIA CON PC	
Indiretto di memoria con PC postindicizzato	EA = [<bd>+PC]+(Xn)+<od>]
Indiretto di memoria con PC preindicizzato	EA = [<bd>+PC+Xn]+<od>

Tab. 5.7 *Le modalità d'indirizzamento dell'MC68020. (continuazione)*

Modalità	**Calcolo dell'indirizzo effettivo**
INDIRIZZAMENTO DI DATI IMMEDIATO	
Immediato	DATI = word successiva/e alla word di operazione
Immediato rapido	Dati inerenti
INDIRIZZAMENTO IMPLICITO	
Registro implicito	EA = SR, USP, SSP, o PC

Note:

1.
EA Indirizzo effettivo
An Registro d'indirizzo
Dn Registro di dati
Xn Registro d'indirizzo o di dati usato come registro indice
SR Registro di stato
PC Contatore di programma
<d8> Offset (spostamento) di 8 bit
<d16> Offset (spostamento) di 16 bit
<bd> Valore di 16 o di 32 bit (spostamento di base)
<od> Valore di 16 o di 32 bit (spostamento esteriore)
[ ] Indiretto di memoria
( ) Contenuto di
← Sostituisce

2. Nelle modalità d'indirizzamento di postincremento e di predecremento, N = 1 per byte, N = 2 per word, N = 4 per longword. Se An è il puntatore di stack di sistema e la dimensione dell'operando è in byte, dev'essere N = 2 per tenere il puntatore di stack su un confine di word.

3. Xn può essere scalato di 1, 2, 4 o 8.

4. Entrambe le forme immediate sono considerate come una sola modalità. L'indirizzamento implicito non è considerato una modalità distinta.

La classificazione degli indirizzi in base alla modalità è comoda per il programmatore, al fine di determinare i modi in cui si può far riferimento agli operandi. Certe modalità d'indirizzamento sono ammesse ed altre sono invece proibite per certe istruzioni specifiche. La letteratura tecnica della Motorola classifica ulteriormente in *categorie* le modalità d'indirizzamento, per semplificare la discussione di quelle modalità disponibili per un'istruzione. Come definito nella Tab. 5.8, le categorie d'indirizzamento sono: *dati*, *memoria*, *controllo*, e *alterabile*; esse si riferiscono alle caratteristiche dell'operando.

Tab. 5.8 *Categorie d'indirizzamento e sintassi dell'assemblatore.*

Modalità d'indirizzamento	Modo	Registro	Dati	Memoria	Controllo	Alterabile	Sintassi dell'assemblatore
Diretto di registro di dati	000	N. reg.	x	—	—	x	Dn
Diretto di registro d'indirizzo	001	N. reg.	—	—	—	x	An
Indiretto di registro d'indirizzo	010	N. reg.	x	x	x	x	(An)
Indiretto di registro d'indirizzo con postincremento	011	N. reg.	x	x	—	x	(An)+
Indiretto di registro d'indirizzo con predecremento	100	N. reg.	x	x	—	x	–(An)
Indiretto di registro d'indirizzo con spostamento	101	N. reg.	x	x	x	x	(d_{16},An)
Indiretto di registro d'indirizzo con indice (spostamento di 8 bit)	110	N. reg.	x	x	x	x	(d_8,An,Xn)
Indiretto di registro d'indirizzo con indice (spostamento di base)	110	N. reg.	x	x	x	x	(bd,An,Xn)
Indiretto di memoria postindicizzato	110	N. reg.	x	x	x	x	([bd,An],Xn,od)
Indiretto di memoria preindicizzato	110	N. reg.	x	x	x	x	([bd,An],Xn,od)
Assoluto corto	111	000	x	x	x	x	(xxx).W
Assoluto lungo	111	001	x	x	x	x	(xxx).W
Indiretto di contatore di programma con spostamento	111	010	x	x	x	—	(d_{16},PC)
Indiretto di contatore di programma con indice (spostamento di 8 bit)	111	011	x	x	x	—	(d_8,PC,Xn)
Indiretto di contatore di programma con indice (spostamento di base)	111	011	x	x	x	—	(bd, PC,Xn)
Indiretto di memoria di PC postindicizzato	111	011	x	x	x	—	([bd,PC],Xn],od)
Indiretto di memoria di PC preindicizzato	111	011	x	x	x	—	([bd,PC],Xn],od)
Immediato	111	100	x	x	—	—	#<dato>

Fonte: Motorola, Inc.

Nella precedente tabella viene elencata anche la sintassi dell'assemblatore per ciascuna modalità d'indirizzamento e la codifica dell'indirizzo effettivo (modo/registro) usata come parte del formato di linguaggio-macchina. Nel sottoparagrafo finale saranno spiegate queste categorie d'indirizzamento e sarà presentata la forma di linguaggio-macchina delle word di estensione richieste da alcune modalità d'indirizzamento.

5.3.1 Indirizzamento diretto di registro e indirizzamento assoluto

Sia nella modalità d'indirizzamento diretto di registro che in quella d'indirizzamento assoluto, la locazione di un operando è specificata esplicitamente nell'istruzione. Un *indirizzo diretto* dell'MC68020 deve far riferimento ad uno degli indirizzi di dati o d'indirizzo del processore. Il contenuto del registro è l'operando. Un *indirizzo assoluto* è un indirizzo di memoria di 16 o di 32 bit; in questo caso, il contenuto della locazione è l'operando. Un indirizzo di 16 bit è definito "corto", mentre un indirizzo di 32 bit è considerato "lungo".

Indirizzamento diretto di registro. In effetti i registri del processore rappresentano una memoria ad alta velocità entro la CPU e le operazioni tra i registri non richiedono alcun riferimento alla memoria esterna. I due modi dell'indirizzamento diretto di registro sono *diretto di registro di dati* e *diretto di registro d'indirizzo.* L'indirizzo effettivo calcolato è:

$$EA = Dn$$

che specifica il registro di dati Dn. Nel modo diretto di registro d'indirizzo, l'indirizzo effettivo è calcolato come segue:

$$EA = An$$

dove An specifica l'n-esimo registro d'indirizzo. L'istruzione che fa riferimento ad un registro di dati può specificare una lunghezza di un byte (B), di una word (W) o di una longword (L) per l'operando. Quindi l'istruzione

```
CLR.<X>    D2
```

azzera il registro D2 per la lunghezza <X> = B o W o L. Questo è il formato tipico per un'istruzione a singolo indirizzo che impiega un registro di dati come destinazione. Un registro d'indirizzo è considerato contenente un indirizzo lungo 16 o 32 bit. Le istruzioni non sono disponibili per operazioni di byte che impiegano registri d'indirizzo. Per esempio, l'istruzione:

```
MOVE.<X>   A1,D1
```

consentirà di specificare una dimensione lunga soltanto <X> = W o <X> = L. Oltre al vincolo sulla dimensione, altre limitazioni sono poste sull'impiego del registro d'indirizzo come locazione di destinazione.

L'MC68020 consente il trattamento di indirizzi mediante un gruppo di istruzioni che utilizzano un registro d'indirizzo come destinazione. Queste istruzioni normalmente sono contrassegnate dal suffisso "A" nella notazione in linguaggio assembler. Per esempio, l'istruzione

```
MOVEA.L    A1,A2
```

trasferisce il contenuto di 32 bit di A1 ad A2. L'istruzione MOVEA (*MOVE Address:* trasferisci indirizzo) svolge la medesima funzione dell'istruzione MOVE, ma è intesa a trattare gli operandi come indirizzi (cioè, come numeri positivi), indicanti locazioni di memoria. Pertanto i registri di dati e i registri d'indirizzo dell'MC68020 non sono considerati equivalenti per molte operazioni. Le differenze saranno presentate in dettaglio nel cap. 9, in cui saranno prese in considerazione le istruzioni che operano su indirizzi.

Indirizzamento assoluto corto. L'indirizzamento corto o di 16 bit per un operando nella memoria è contenuto in una word di estensione per l'istruzione. L'indirizzo viene effettivamente convertito in un indirizzo di 32 bit, estendendo il bit di segno (bit 15 dell'indirizzo corto. La Fig. 5.10(a) illustra il calcolo. Un indirizzo di 32 bit è usato per indirizzare la memoria con sequenze di 0 o 1 di testa nei bit [31:16], replicando il bit più significativo della word di estensione. L'indirizzo esteso è considerato un numero positivo di 32 bit.

Generazione: EA assegnato
Sintassi di assemblatore: (xxx).W
Modo: 111
Registro: 000
Word di estensione: → [31 Esteso di segno | Indiriz. di memoria 0]
Indirizzo di memoria: [31 Operando 0]

(a) *Assoluto corto.*

Generazione: EA assegnato
Sintassi di assemblatore: (xxx).L
Modo: 111
Registro: 001
Prima word di estensione: → [15 Indirizzo alto 0]
Seconda word di estensione: → [15 Indirizzo basso 0]
[31 Concatenazione 0]
Indirizzo di memoria: [31 Operando 0]

(b) *Assoluto lungo.*

Fig. 5.10 *Indirizzamento assoluto. (Per gentile concessione di Motorola, Inc.)*

Esempio 5-7

La Fig. 5.11 mostra l'effetto risultante dall'impiego della modalità d'indirizzamento corto. Se l'indirizzo esadecimale è compreso tra 0 e $7FFF, allora si possono indirizzare i 32 kilobyte bassi della memoria. Gli indirizzi corti maggiori o uguali a $8000 sono estesi di segno a 32 bit, producendo indirizzi di memoria compresi tra $FFFF8000 e $FFFFFFFF, che sono i 32 kilobyte superiori di memoria indirizzabile dell'MC68020. Quindi i progettisti dell'insieme d'istruzioni hanno trattato questi segmenti di memoria come gli estremi in maniera speciale. Normalmente, il progettista di sistema specifica il segmento più basso della memoria per i parametri di sistema. Infatti, il processore MC68020 utilizza i primi 1024 byte per i suoi vettori, che definiscono gli indirizzi iniziali per le routine di trappola e d'interruzione. Il segmento più alto è riservato per le interfacce di I/O in molti sistemi. La modalità d'indirizzamento assoluto corto permette un accesso efficiente a locazioni fisse nell'una o l'altra di queste regioni.

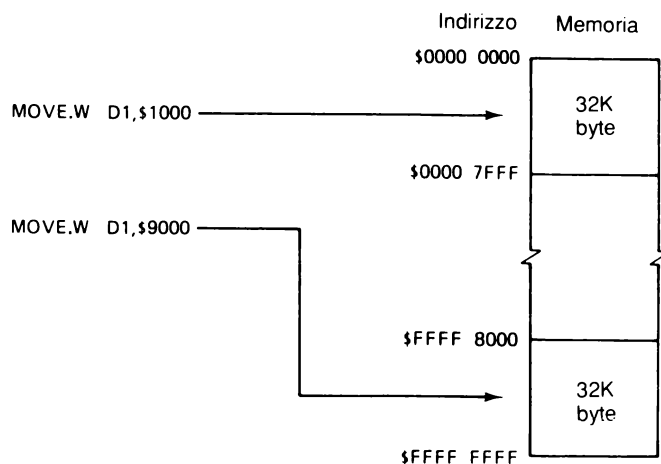



Fig. 5.11 *Esempio d'indirizzamento assoluto corto.*

Indirizzamento assoluto lungo. Come mostrato nella Fig. 5.10(b), un indirizzo lungo è formato da due word di estensione che seguono la word di operazione di un'istruzione. Tale indirizzo può estendersi sull'intero spazio d'indirizzamento della memoria dell'MC68020. La grandezza dell'indirizzo assoluto specificata nell'istruzione del linguaggio assembler determina la modalità d'indirizzamento assoluto — corto o lungo — impiegata. L'istruzione

```
MOVE.W    $12000,D1
```

richiede l'indirizzamento assoluto lungo per l'operando di sorgente poiché l'indirizzo richiede più di 16 bit. Un'istruzione quale:

```
MOVE.B      $3FFF,$12000
```

specifica entrambe le modalità d'indirizzamento. L'indirizzo di sorgente potrebbe essere specificato come $0000 3FFF, che costringerebbe la maggior parte degli assemblatori ad utilizzare il modo assoluto lungo. Consultando il manuale del linguaggio assembler del particolare assemblatore da utilizzare, si determinerà il modo in cui questi indirizzi sono gestiti e convertiti in linguaggio-macchina.

Esempio 5-8

La Fig. 5.12 mostra alcuni modi d'indirizzamento appena discussi, così come potrebbero essere impiegati nelle istruzioni. Nella figura sono elencate anche le istruzioni di linguaggio-macchina così come sono registrate nella memoria. Le istruzioni specificano le modalità di registri come una word di lunghezza e sono le più efficienti per quanto concerne la memorizzazione e l'esecuzione. Come notato in precedenza, il codice mnemonico MOVEA è usato come codice operativo quando la destinazione è un registro d'indirizzo.

Nelle modalità assolute, l'indirizzo è indipendente dalla lunghezza dell'operando. L'istruzione

```
MOVE.L      $0534,$0528
```

copia il valore di 32 bit nelle locazioni da $0534 a $0537 in quattro byte a partire dall'indirizzo $0528, poiché sono specificati operandi di longword. Tuttavia, entrambi gli indirizzi sono assoluti corti.

```
                                 1.          TTL       FIGURA 5.12
                                 2.  *
0'000000  2406                   3.          MOVE.L    D6,D2           ;(D2)=(D6)
0'000002  4240                   4.          CLR.W     D0              ;(D0)[15:0]=0
0'000004  2640                   5.          MOVEA.L   D0,A3           ;(A3)=(D0)
0'000006  1038 2001              6.          MOVE.B    $2001,D0        ;(D0)[7:0]=($2001)
0'00000A  21F9 000214AA          7.          MOVE.L    $214AA,$24      ;($24)=($214AA)
          0024
0'000012  21F8 0534 0528         8.          MOVE.L    $0534,$0528     ;($0528)=($0534)
                                 9.
```

Fig. 5.12 *Esempi di indirizzamento di registro ed assoluto.*

5.3.2 Indirizzamento indiretto di registro

Sono disponibili sei modalità d'indirizzamento indiretto per l'MC68020 per far riferimento ad un operando che fa parte di una struttura di dati nella memoria. Tutte e sei le modalità impiegano un registro d'indirizzo per contenere l'indirizzo

fondamentale, che può quindi essere modificato in vari modi per calcolare l'indirizzo effettivo di un operando specifico. A differenza degli indirizzi assoluti che sono definiti allorché il programma viene assemblato, l'indirizzo indiretto è calcolato mentre il programma viene eseguito. In questo paragrafo sono discussi l'indirizzamento indiretto di registro d'indirizzo, quello indiretto di registro con spostamento e due modi indiretti di registro con indicizzazione. I modi indiretti con postincremento e con predecremento saranno discussi nel sottopar. 5.3.3.

Modo indiretto di registro d'indirizzo. Uno qualunque degli otto registri d'indirizzo dell'MC68020 può essere impiegato per indirizzare indirettamente un operando nella memoria. L'indirizzo effettivo è calcolato come:

EA = (An)

dove è specificato il registro *n*-esimo. Il contenuto del registro d'indirizzo prescelto viene usato come indirizzo di operando allorché viene eseguita un'istruzione che impiega questa modalità. Per caricare il registro, un'istruzione quale:

```
MOVEA.L    #<indir>,A1
```

trasferisce l'indirizzo <indir> in A1. Poi, facendo riferimento ad (A1), come nell'istruzione

```
MOVE.W     (A1),D1
```

il valore di 16 bit contenuto nella locazione < indir> sarà trasferito in D1. L'indirizzo impiegato come sorgente nell'istruzione MOVEA può essere specificato da una qualsiasi delle modalità d'indirizzamento, incluso il modo immediato appena descritto. Per esempio, l'istruzione

```
MOVEA.L    $1000,A1
```

trasferisce in A1 il contenuto della longword alla locazione $1000.

La Fig. 5.13(a) illustra il calcolo dell'indirizzo di operando in questa modalità indiretta. L'assemblatore riconosce l'indirizzamento indiretto quando (An) è usato per designare un operando di sorgente o di destinazione. In questo testo, (An) denota il contenuto del registro An, mentre ((An)) indica l'operando, cioè il contenuto del contenuto di An.

Esempio 5-9

L'indirizzo contenuto in un registro d'indirizzo può essere modificato in un certo numero di modi durante l'esecuzione del programma. Per esempio, l'istruzione di somma degli indirizzi (*ADD Address:* ADDA) ha il formato illustrato a pagina seguente.

Generazione: EA = (An)
Sintassi di assemblatore: (An)
Modo: 010
Registro: n
Registro d'indirizzo: An → 31 Indirizzo di memoria 0
Indirizzo di memoria: 31 Operando 0

(a) *Indiretto di registro d'indirizzo.*

Generazione: EA = (An) + d_{16}
Sintassi di assemblatore: (d_{16},An)
Modo: 101
Registro: n
Registro d'indirizzo: An → 31 Indirizzo di memoria 0
Spostamento: 31 Esteso di segno 15 Intero 0 → (+)
Indirizzo di memoria: 31 Operando 0

(b) *Diretto di registro d'indirizzo con spostamento di 16 bit.*

Fig. 5.13 *Indirizzamento indiretto di registro. (Per gentile concessione di Motorola, Inc.)*

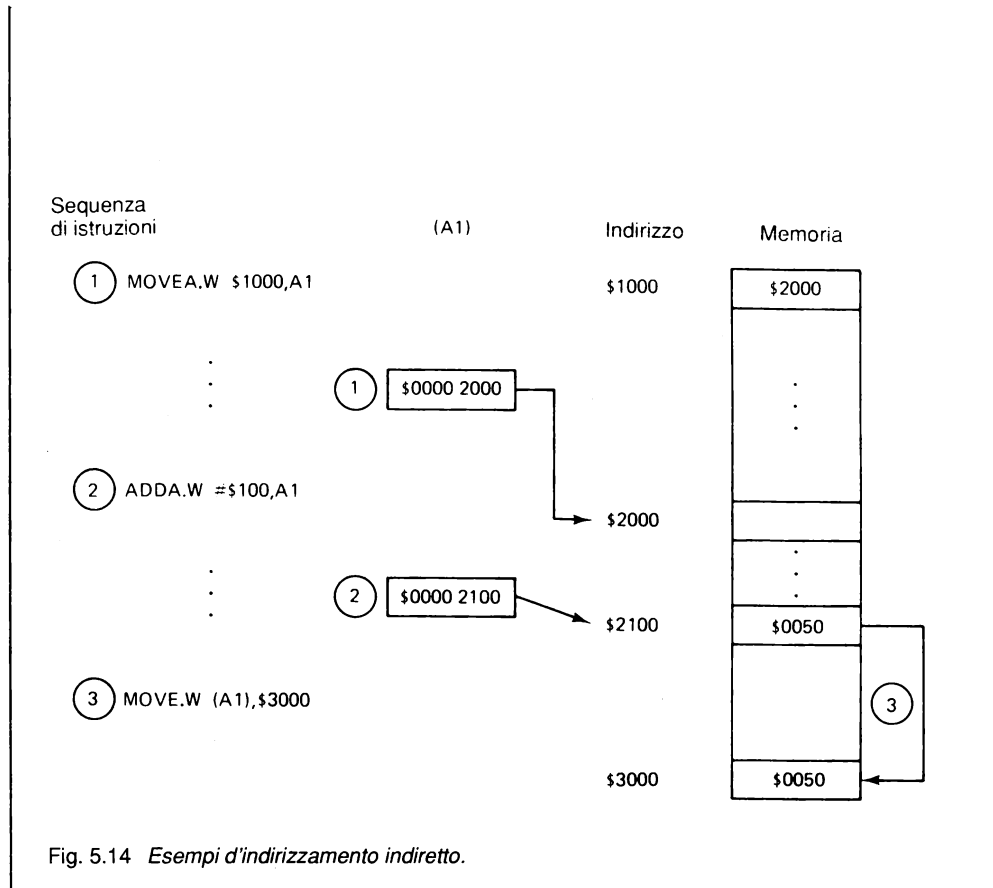
ADDA.<X> <EAs>,<An>

per sommare il valore nella locazione <EAs> al valore in An. La lunghezza è ristretta a W o L per valori di 16 e di 32 bit, rispettivamente. L'indirizzo

<An> ← <An> + <EAs>

viene generato quando l'istruzione ADDA viene eseguita. Il riferimento tramite An ad un operando in un'istruzione successiva produrrebbe l'indirizzamento di una nuova locazione. La Fig. 5.14 illustra una possibile sequenza.

Dapprima viene trasferito in A1 il valore $2000, che rappresenta il contenuto della locazione $1000. Questo indirizzo $2000 potrebbe puntare alla prima word in una struttura di dati nella memoria. L'aggiunta della costante $100 usando la variante ADDA di ADD fa sì che A1 punti alla locazione $2100, cioè un offset di 100 locazioni (in esadecimale) dall'inizio della struttura di dati. L'impiego indiretto di A1 nell'istruzione MOVE causa il trasferimento del valore contenuto nella locazione $2100 alla locazione designata in $3000.

Fig. 5.14 *Esempi d'indirizzamento indiretto.*

Modo indiretto con spostamento. Il modo indiretto di registro d'indirizzo, appena discusso, può essere usato in varie maniere per indirizzare operandi nella memoria. Usando le istruzioni di trattamento degli indirizzi dell'MC68020 (MOVEA, ADDA, ed altre), l'indirizzo nel registro An può essere modificato come desiderato durante l'esecuzione del programma. In certe applicazioni, non è opportuno modificare il contenuto del registro d'indirizzo. Questo caso potrebbe presentarsi quando il registro viene usato per puntare ad un segmento di memoria, magari il primo indirizzo di un array o di una tabella. Un elemento selezionato nella struttura di dati potrebbe essere indirizzato sommando un offset al valore nel registro d'indirizzo. Questo registro conterrebbe allora l'indirizzo di partenza, mentre l'offset o spostamento specifica la posizione relativa dell'elemento entro la struttura di dati. Quando l'offset è un valore fissato che risulta noto in fase di assemblaggio, si può usare il modo d'indirizzamento *indiretto con spostamento* dell'MC68020 per far riferimento all'operando.

Come mostrato nella Fig. 5.13(b), l'indirizzo effettivo per questa modalità è calcolato dall'equazione:

EA = (An) + <d_{16}>

dove <d_{16}> è un intero con segno di 16 bit.[6] L'intervallo di questo offset di 16 bit è da –32768 a +32767 byte nella memoria. Questo spostamento o offset è memorizzato come una word di estensione dell'istruzione in linguaggio-macchina. L'assemblatore impiega la modalità d'indirizzamento indiretto con spostamento in istruzioni quali:

```
MOVE.W    <d16>(An),<EAd>
```

dove <d_{16}> è specificato come una costante esadecimale ($XXXX) o decimale (XXXX). Per esempio:

```
MOVE.W    $20(A1),D1
```

trasferisce 16 bit al registro D1 dalla locazione situata 32 byte oltre l'indirizzo in A1. Si noti che il simbolo immediato "#" non è necessario, poiché l'assemblatore riconosce <d_{16}> come un offset e non come un possibile indirizzo. La forma con lo spostamento racchiuso tra parentesi, cioè:

```
MOVE.W    ($20,A1),D1
```

è equivalente. Tuttavia, questa forma del linguaggio assembler consente spostamenti di 8, 16 o 32 bit, come sarà descritto nel prossimo paragrafo. Si tratta di un modo indiretto di registro d'indirizzo in cui il registro indice è soppresso.

Modo indiretto con indicizzazione, scalamento e spostamento di base. Quando l'offset di un indirizzo di base dev'essere variato durante l'esecuzione del programma, si può impiegare uno dei due modi indiretti di registro con indicizzazione. L'indirizzo effettivo viene calcolato come la somma di tre componenti: un valore in un registro d'indirizzo, più un valore in un registro indice, più uno spostamento. Il registro indice può essere uno qualsiasi dei registri generali d'indirizzo o di dati dell'MC68020.

Il calcolo fondamentale per l'indirizzo effettivo avviene come segue:

EA = (An) + (Xi).[S]*SCALA + <spostamento>

dove Xi è il registro indice designato da uno qualsiasi dei registri An o Dn. Esso può essere designato come un valore di 16 bit o di 32 bit nel registro indice. Quindi la designazione Xi.[S] consente [S] = W o L per word o longword, rispettivamente.

---

[6] In questo contesto, il deponente denota la lunghezza dell'intero, *non* la base di numerazione.

Qualsiasi valore dell'indice di 16 bit viene sempre esteso di segno a 32 bit quando è impiegato nel calcolo di un indirizzo. Se il valore nel registro indice è un numero esadecimale da $8000 a $FFFF, allora tale valore è considerato negativo. Il valore di scala (SCALA) consente una variazione di scala di 1 (nessuno scalamento), o di 2, 4 o 8. Con un valore di SCALA = n, il registro indice effettua l'indicizzazione per n byte. Quindi il valore SCALA = 2 indica l'indirizzamento di word, mentre il valore SCALA = 4 denota l'indirizzamento di longword. Il valore dello spostamento può essere di 8, 16 o 32 bit. L'indice con una modalità di spostamento di 8 bit è mostrato nella Fig. 5.15(a) e definisce il calcolo dell'indirizzo effettivo come segue:

$$E = (An) + (Xi).[S]*SCALA + <d_8>$$

dove $<d_8>$ è un intero con segno di 8 bit.[7] La seguente istruzione del linguaggio assembler:

```
MOVE.W    2(A1, D1.W), D2
```

trasferisce 16 bit dall'indirizzo specificato da:

$$EA = (A1) + (D1)[15:0] + 2$$

al registro (D2)[15:0]. Questa modalità richiede una sola word di estensione all'istruzione nella memoria.

La Fig. 5.15(b) illustra il calcolo dell'indirizzo effettivo per l'indirizzamento indiretto di registro con indice e spostamento di base. Ciò differisce dalla modalità precedente in quanto lo spostamento di base è un valore di 16 bit (esteso di segno) o di 32 bit. Inoltre, tutti e tre gli addendi specificati sono facoltativi. Quindi l'istruzione:

```
MOVE.L    ($900, A3, D2.W*4), D1
```

con (A3) = $00020000 e (D2) = $00001000 indirizza la locazione esadecimale

20000	base
4000	indice, scalato
900	spostamento di base
24900	

e trasferisce 32 bit in D1 da quella locazione. Un'istruzione può richiedere una, due o tre word di estensione nella memoria, a seconda della lunghezza dello spostamento di base e delle opzioni selezionate. Nell'esempio appena illustrato, sono richieste due word di estensione per specificare la forma d'indirizzamento e lo spostamento di 16 bit.

[7] Lo spostamento di 8 bit è incluso nell'insieme di istruzioni delll'MC68020 per la compatibilità con l'MC68000. Lo spostamento di base di 16 o di 32 bit non era ammesso dai modi d'indirizzamento dell'MC68000.

Generazione: $EA = (An) + (Xn) + d_8$
Sintassi di assemblatore: (d_8,An,Xn.DIM * SCALA)
Modo: 110
Registro: n
Registro d'indirizzo: An → Indirizzo di memoria (31 … 0)

Spostamento: Esteso di segno (31) | Intero (7 … 0)
Registro indice: Valore esteso di segno (31 … 0)
Scala: Valore di scala
Indirizzo di memoria: Operando (31 … 0)

(a) *Indiretto di registro d'indirizzo con indice (8 bit).*

Generazione: $EA = (An) + (Xn) + bd$
Sintassi di assemblatore: (bd,An,Xn.DIM * SCALA)
Modo: 110
Registro: n
Registro d'indirizzo: An → Indirizzo di memoria (31 … 0)

Spostamento di base: Valore esteso di segno (31 … 0)
Registro indice: Valore esteso di segno (31 … 0)
Scala: Valore di scala
Indirizzo di memoria: Operando (31 … 0)

(b) *Indiretto di registro d'indirizzo con indice e spostamento di base.*

Fig. 5.15 *Indirizzamento indiretto di registro d'indirizzo con indice. (Per gentile concessione di Motorola, Inc.)*

Esempio 5-10

Si supponga che una lista di word sia contenuta nella memoria a partire dalla locazione $10000. Se i primi quattro elementi di 16 bit nella lista sono usati per determinare la lunghezza della lista e simili informazioni, l'accesso al primo elemento di dati può avvenire sommando uno spostamento di 8 byte

all'indirizzo iniziale. L'accesso ad elementi consecutivi nella lista può essere effettuato mediante indicizzazione e scalamento. Questo metodo è mostrato nella Fig. 5.16, impiegando A1 come indirizzo di base e D1 come registro indice. La prima esecuzione dell'istruzione all'etichetta LOOP indirizza la locazione $10008.

```
                          1.        TTL      FIGURA 5.16
                          2.  *
0'000000  227C 00010000   3.        MOVEA.L #$10000,A1         ;INIZIO DELLA LISTA
0'000006  4281            4.        CLR.L   D1                 ;INDICE ZERO
                          5.  *
                          6.  *
                          7.  *
0'000008  3431 1A08       8.  LOOP  MOVE:W  (8,A1,D1.L*2),D2   ;TRASFERISCE ELEMENTO IN D2
                          9.                                   ; SCALA INDICE
                         10.                                   ; PER WORD
                         11.  *
                         12.  *
                         13.  *
0'00000C  5241           14.        ADD.W   #1,D1              ;WORD SUCCESSIVA
                         15.  *     (SALTA A LOOP SE NON HA FINITO)
                         16.  *
0'00000E                 17.        END
```

Indirizzo	Contenuto		Indirizzamento
$10000	Intestazione	← (A1)	:1
$10008	Elem. 1	← (A1) + 8	:2
$1000A	Elem. 2		
.	⋮		
.		← (A1) + 8 + (D1)	:*k*-mo ciclo
.	⋮		

(b) *Riferimento di memoria.*

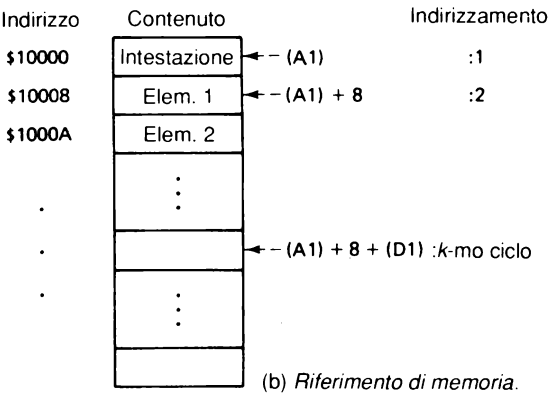



Fig. 5.16 *Esempio di indirizzamento indiretto con indicizzazione e scalamento.*

5.3.3 Indirizzamento con predecremento e postincremento

In molte applicazioni di programmazione, è necessario accedere a dati contenuti in locazioni di memoria consecutive. Se la lunghezza di ciascun elemento è di un byte, di una word o di una longword, l'indicizzazione può essere effettuata mediante l'indirizzamento *indiretto con predecremento* o *indiretto con postincremento*

dell'MC68020. Il contenuto di un registro d'indirizzo viene modificato automaticamente durante l'esecuzione delle istruzioni che impiegano tali modalità. Quindi non c'è lo spreco di tempo che si avrebbe se le istruzioni del programma dovessero incrementare i valori degli indici, come avviene nella modalità d'indirizzamento indiretto con indicizzazione. Forse l'uso più importante di queste modalità è quello del trattamento degli operandi in uno stack della memoria, come discusso nel cap. 4. Comunque, l'impiego di queste modalità semplifica molte altre operazioni di trattamento dei dati.

L'indirizzo effettivo per ciascuna modalità viene calcolato come mostrato nella Fig. 5.17. Nel modo con postincremento, l'indirizzo in An viene usato prima che An venga incrementato di 1, 2 o 4 byte per operandi di byte, word o longword, rispettivamente. Questa modalità consente ad un programma d'indirizzare valori consecutivi memorizzati ad indirizzi sempre maggiori nella memoria. Nella modalità con predecremento, An viene dapprima incrementato e poi usato come puntatore ad una locazione di memoria.

Generazione: EA = (An)
An = An + DIM
Sintassi di assemblatore: (An)+
Modo: 011
Registro: n
Registro d'indirizzo: An
31 0
Indirizzo di memoria
Lunghezza di operando (1, 2 o 4):
+
31 0
Indirizzo di memoria: Operando

(a) *Postincremento.*

Generazione: An = An − DIM
EA = (An)
Sintassi di assemblatore: −(An)
Modo: 100
Registro: n
Registro d'indirizzo: An
31 0
Indirizzo di memoria
Lunghezza di operando (1, 2 o 4):
−
31 0
Indirizzo di memoria: Operando

(b) *Predecremento.*

Fig. 5.17 *Indirizzamento con predecremento e con postincremento. (Per gentile concessione di Motorola, Inc.)*

L'assemblatore riconosce come –(An) il modo con predecremento e come (An)+ il modo con postincremento. Nell'istruzione

```
MOVE.W    -(A1),(A2)+
```

una word viene trasferita dalla locazione (A1)–2 alla locazione (A2). Se l'istruzione viene eseguita nuovamente, l'indirizzo di sorgente si trova una word più in basso nella memoria, mentre la destinazione è situata una word più in alto rispetto ai valori originali. Questo metodo è solitamente impiegato in cicli di programma in cui l'istruzione autoindicizzata viene eseguita ripetutamente finché non si verifica la condizione per uscire dal ciclo.

Queste modalità sono utilizzate anche per il trasferimento di blocchi di dati da un segmento di memoria ad un altro, quando in un ciclo vengono usate istruzioni come la seguente:

```
MOVE.W    (A1)+,(A2)+
```

Qui (A1) designa il primo blocco e (A2) il secondo. L'indirizzamento in questo caso ha un effetto equivalente a quello della sequenza di istruzioni:

```
MOVE.W    (A1),(A2)
ADD.L     #2,A1
ADD.L     #2,A2
```

dove sia l'indirizzo di sorgente che quello di destinazione sono incrementati di 2 dopo il trasferimento, poiché gli operandi hanno una lunghezza di due byte.

Esempio 5-11

Il semplice segmento di programma nella Fig. 5.18 effettua il trasferimento di un blocco di 32 byte di dati dalla locazione $20000 alla locazione $30000, ma in ordine inverso. L'ordine dei dati viene rovesciato iniziando il trasferimento dal primo byte del blocco di sorgente mediante l'indirizzamento con postincremento, fino all'ultimo byte del blocco di destinazione mediante indirizzamento con predecremento. Il byte in $20000 viene trasferito a $3001F, il byte in $2001 viene trasferito in $3001E, e così via. Il registro D1 viene usato come un contatore per il ciclo e i registri A1 e A2 contengono gli indirizzi dei blocchi.

5.3.4 Indirizzamento indiretto di memoria

Nella modalità d'indirizzamento assoluto, l'indirizzo specificato con la locazione di memoria conteneva l'operando. L'indirizzamento indiretto di registro consente che l'indirizzo sia contenuto in un registro d'indirizzo per specificare la locazione

```
                         1.          TTL      FIGURA 5.18
                         2.
                         3.          LLEN     100
00010000                 4.          ORG      $10000
                         5.  *
  = 00000063             6.  RETURN  EQU      $0063          ;MONITOR
                         7.  *
                         8.  *       TRASFERISCE 32 BYTE DA $20000 A $30000
                         9.  *       ROVESCIANDONE L'ORDINE
                        10.  *
00010000 123C 0020      11.          MOVE.B   #32,D1         ;FISSA IL CONTATORE A 32
00010004 227C 00020000  12.          MOVEA.L  #$20000,A1     ;PREDISPONE GLI INDIRIZZI
0001000A 247C 00030020  13.          MOVEA.L  #$30020,A2     ; PER IL TRASFERIMENTO
                        14.  *
00010010 1519           15.  LOOP    MOVE.B   (A1)+,-(A2)    ;TRASFERISCE IL BYTE SUCCESSIVO
00010012 0401 0001      16.          SUBI.B   #1,D1          ;DECREMENTA IL CONTEGGIO
00010016 66   F8        17.          BNE      LOOP           ;CONTINUA FINCHE'
                        18.  *                               ; CONTEGGIO = 0
                        19.  *
00010018 4E4F           20.          TRAP     #15            ;RITORNA AL MONITOR
0001001A 0063           21.          DC.W     RETURN
0001001C                22.          END
```

Fig. 5.18 *Esempio di indirizzamento con postincremento e predecremento. (Esempio 5-11)*

di memoria di un operando. Tale registro, eventualmente modificato mediante indicizzazione e aggiunta di uno spostamento, agisce da *puntatore* all'operando. Le modalità di *indirizzamento indiretto di memoria* dell'MC68020 consentono che un indirizzo contenuto nella memoria punti ad un operando. Questo indirizzo nella memoria è un valore di 32 bit che viene usato come una componente nel calcolo dell'indirizzo effettivo. La Fig. 5.19 illustra un confronto di queste varie modalità d'indirizzamento. In tale figura, il più semplice modo d'indirizzamento indiretto della memoria è fornito nell'istruzione:

```
CLR.W    ([A2)]
```

dove [A2] indica che l'indirizzo è situato nella memoria. Quindi [A2] specifica il contenuto del contenuto di A2 come indirizzo dell'operando. In altre parole, l'istruzione causa l'azzeramento della locazione con indirizzo ((A2)).

Il modo d'indirizzamento assoluto non consente che un indirizzo sia modificato durante l'esecuzione del programma senza modificare la word di estensione dell'istruzione stessa nella memoria. Poiché ciò avviene raramente, l'indirizzo è considerato fisso allorché il programma viene assemblato. Il modo indiretto di registro è più flessibile, ma l'MC68020 ha soltanto sette registri d'indirizzo di finalità generale. Qualora un programma richiedesse un maggior numero di indirizzi indiretti, essi potrebbero venire registrati nella memoria e quindi trasferiti al registro d'indirizzo appropriato nel momento in cui se ne presenti la necessità. La sequenza di istruzioni che impiegano l'indirizzamento indiretto di registro pe azzerare la locazione $10000 potrebbe essere la seguente:

```
MOVE.L    $20000,A1      ;(A1) ← (locazione $20000)
CLR.W     (A1)
```

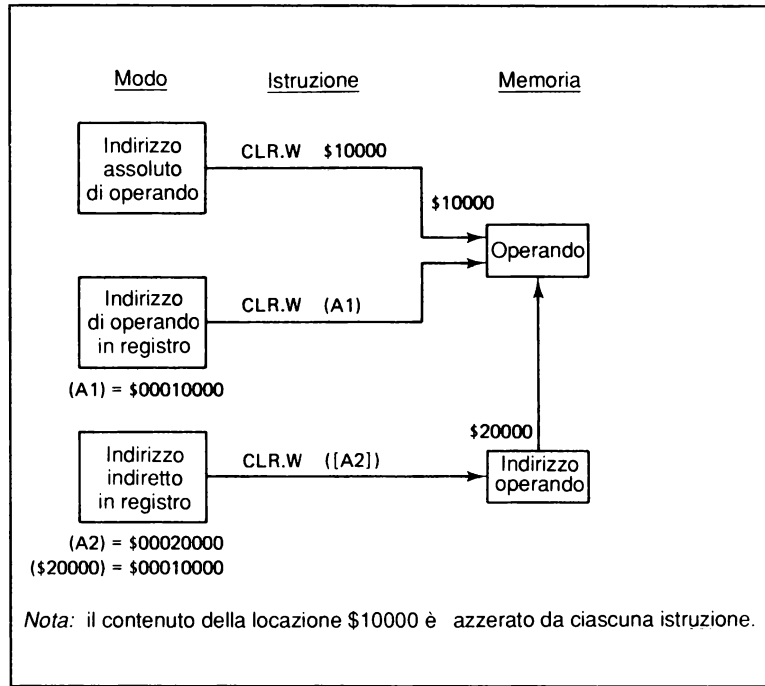
Fig. 5.19
Un confronto tra le varie modalità di indirizzamento.

Modo | Istruzione | Memoria

Indirizzo assoluto di operando — CLR.W $10000 — $10000 → Operando

Indirizzo di operando in registro — CLR.W (A1) → Operando
(A1) = $00010000

Indirizzo indiretto in registro — CLR.W ([A2]) → Indirizzo operando — $20000
(A2) = $00020000
($20000) = $00010000

Nota: il contenuto della locazione $10000 è azzerato da ciascuna istruzione.

Questa sequenza azzera la locazione puntata dall'indirizzo contenuto nella locazione $20000. Se ($20000) = $10000, allora viene azzerato il valore lungo una word alla locazione $10000. Il medesimo risultato è ottenibile con la seguente istruzione dell'MC68020:

```
CLR.W      [(A2)]
```

se A2 contiene $20000. Un ovvio vantaggio del modo indiretto di memoria è che il numero di indirizzi indiretti (contenuti nella memoria) è praticamente illimitato se A2 viene modificato per puntare a distinte locazioni di memoria. Ciò si ottiene facilmente tramite l'indicizzazione.

Nell'MC68020 sono possibili due modalità d'indirizzamento indiretto di memoria che impiegano i registri d'indirizzo. La prima è definita *postindicizzata*: tale attributo deriva dal fatto che un valore dell'indice viene considerato aggiunto all'indirizzo indiretto nella memoria dopo che ha avuto luogo l'indirizzamento indiretto. La seconda modalità è quella definita *preindicizzata*: in questo caso, il valore dell'indice viene usato per calcolare l'indirizzo della locazione che contiene l'indirizzo indiretto nella memoria. Entrambe queste modalità si rivelano maggiormente utili quando si utilizzano una o più tabelle di indirizzi nella memoria. Questi indirizzi, a loro volta, puntano ad una o più tabelle di operandi. Comunque, tutti i modificatori d'indirizzo specificati, incluso il registro d'indirizzo di base, sono facoltativi. Pertanto il programmatore può specificare esattamente la maniera in cui dev'essere calcolato l'indirizzo indiretto.

Indirizzamento indiretto di memoria postindicizzato. La Fig. 5.20 illustra il calcolo dell'indirizzo effettivo nella modalità d'indirizzamento indiretto della memoria postindicizzato. Il calcolo è espresso simbolicamente come segue:

EA = [<bd> + An] + (Xn).[S]*SCALA + <od>

dove la locazione di memoria che contiene l'indirizzo indiretto si ottiene sommando lo spostamento della base <bd> ad (An). Tale valore viene impiegato come un indirizzo di base per gli operandi di memoria, ognuno dei quali è individuato sommando il valore dell'indice e lo spostamento esteriore o esterno (*outer displacement*) <od>. Come per altre modalità d'indirizzamento, lo spostamento di base <bd> e lo spostamento esteriore <od> sono valori di 16 bit (estesi di segno) o di 32 bit. Il registro indice è trattato come una word estesa di segno ([S] = W) o come un valore di 32 bit ([S] = L) e può essere scalato di 1 (nessuno scalamento), di 2, di 4 o di 8.

Generazione: EA = (bd + An) + Xn.DIM * SCALA + od
Sintassi di assemblatore: ([bd,An,Xn.DIM * SCALA],od)
Modo: 110
Registro d'indirizzo: An → 31 Indirizzo di memoria 0
Spostamento di base: 31 Valore esteso di segno 0
31 Indirizzo di memoria indiretto 0
Punta a
31 Valore all'indirizzo di memoria indiretto 0
Registro indice: 31 Valore esteso di segno 0
Scala: Valore di scala
Spostamento esteriore: 31 Valore esteso di segno 0
Indirizzo effettivo: 31 Operando 0

Fig. 5.20 *Indirizzamento indiretto di memoria postindicizzato. (Per gentile concessione di Motorola, Inc.)*

Di solito, la specificazione minima sarebbe un registro d'indirizzo in un'istruzione. Quindi l'istruzione

```
JMP        ([A1])
```

causerebbe il trasferimento del controllo all'indirizzo specificato nella locazione puntata da A1. Se A1 fosse modificato per puntare ad un altro indirizzo, la word di controllo passerebbe alla routine specificata nella nuova locazione allorché l'istruzione JMP viene rieseguita. Tale uso dell'indirizzamento indiretto di memoria per controllare l'attività di un programma conduce al concetto di "tabella di salto" (*jump table*) o "tabella di distribuzione" (*dispatch table*), come si discuterà nel cap. 6.

Se la nuova routine avesse un certo numero di punti di entrata che erano stati selezionati incrementando un registro indice, allora l'istruzione JMP potrebbe essere modificata come segue:

```
JMP        ([A1]),D1.L*4)
```

dove [A1] punta all'inizio della routine. D1 selezionerebbe l'indirizzo di partenza appropriato entro la routine. Esso è scalato di 4 per effettuare l'indirizzamento su confini di longword. Quindi, se [A1] = $10000 e (D1) = 4, l'indirizzo iniziale del quarto punto di entrata sarebbe $10000, cioè 4 longword o 16 byte dall'indirizzo della prima entrata.

Esempio 5-12

Si consideri la struttura di dati in Fig. 5.21 che mostra una tabella con indirizzo iniziale ADDRTAB = $20000. Un indirizzo situato dopo l'informazione d'intestazione punta alle entrate di operandi in un'altra tabella avente l'indirizzo OPRTAB = $20100. Si supponga che la tabella in ADDRTAB abbia un'intestazione di otto byte seguita dall'indirizzo della tabella di operandi. Il programma nella Fig. 5.22 è progettato per trasferire il secondo byte di ciascuno dei 10 operando di quattro byte alla tabella definita da BYTETAB = $20500. Dopo che i registri sono stati inizializzati in fase di esecuzione del programma, A0 punta all'inizio della tabella d'indirizzi, mentre A1 punta alla tabella di byte.

L'indirizzo della tabella di operandi di longword è specificato come (A0) + 8 nell'istruzione MOVE.B etichettata come LOOP. Ciascun byte da trasferire dalla tabella che inizia in OPRTAB ha un offset di +1 dall'inizio di ciascun operando di longword in questa tabella. Questi operandi di longword sono indirizzati in sequenza incrementando D0, scalato di 4 ogni volta che viene eseguita l'istruzione di trasferimento (MOVE). L'indirizzo della tabella di byte BYTETAB per i byte trasferiti è contenuto in A1, che viene postincrementato di 1 dopo ogni trasferimento. D1 viene decrementato da 10 fino a 0 per controllare il numero di trasferimenti avvenuti. Quando (D1) = 0, sono stati eseguiti 10 cicli ed il programma restituisce il controllo al monitor.

L'istruzione MOVE.B impiega l'indirizzamento indiretto di memoria con postindicizzazione per indirizzare i byte negli operandi di longword. Lo spostamento esteriore di +1 seleziona il secondo byte di ciascuno di questi operandi. In questo esempio, gli indirizzi delle varie tabelle sono definiti con direttive EQU. La lunghezza della tabella di operandi è fissata a 10. Questi valori non possono essere modificati senza riassemblare il programma, dopo che i valori sono stati modificati. L'operando di longword dev'essere memorizzato nella tabella a partire da OPRTAB, prima che il programma sia eseguito. L'informazione d'intestazione non è utilizzata in questo esempio.

31 0
(A0) → INTESTAZIONE
$20000 + 8 → $20100
Tabella di indirizzi (ADDRTAB)
$20100 → VAL1
VAL2
VAL10
Tabella di operandi di longword (OPRTAB)
(A1) = $20500 → VAL1 VAL2 VAL3 VAL4
VAL5 VAL6 VAL7 VAL8
VAL9 VAL10
Tabella di valori (lunghezza di byte) (BYTETAB)
Memoria (longword)

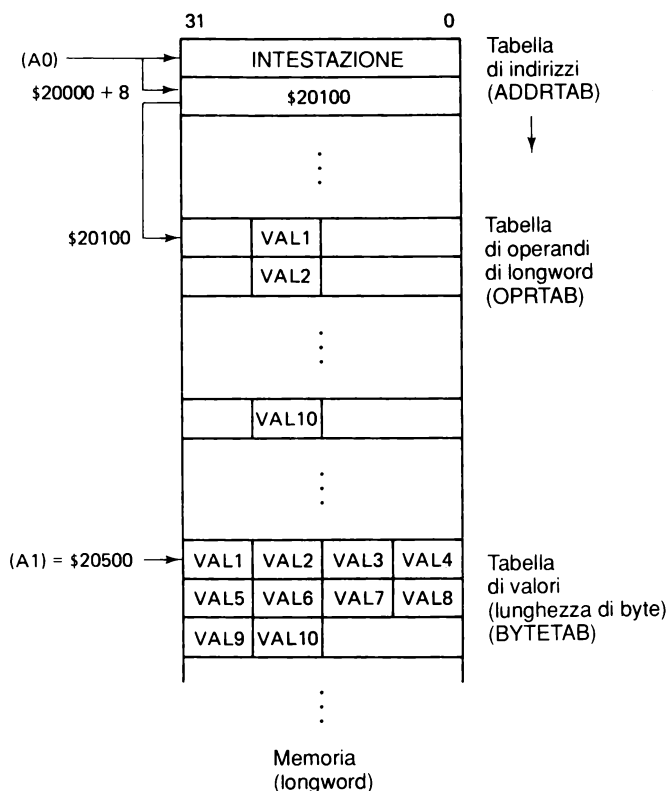



Fig. 5.21 *Struttura di dati nella memoria per indirizzamento indiretto di memoria postindicizzato.*

```
                             1.           TTL      FIGURA 5.22
                             2.           LLEN     100
00010000                     3.           ORG      $10000
                             4.  *
                             5.  *        TRASFERISCE 10 BYTE DA UNA TABELLA DI OPERANDI DI LONGWORD
                             6.  *        ALLA LOCAZIONE $20100. I BYTE HANNO UN OFFSET DI +1
                             7.  *        IN OGNI OPERANDO. LA TABELLA DI BYTE E' IN $20500.
                             8.  *
                             9.  *        L'INDIRIZZO DEL PRIMO OPERANDO DI LONGWORD E' IN UNA
                            10.  *        TABELLA ALLA LOCAZIONE $20000, CON OFFSET DI +8.
                            11.  *
                            12.  *        (A0)+8 CONTIENE L'INDIRIZZO DELLA TABELLA DI OPERANDI.
                            13.  *        (A1) CONTIENE L'INDIRIZZO DELLA TABELLA DI BYTE.
                            14.  *
      # 00020000            15.  ADDRTAB EQU      $20000               ;TABELA DI INDIRIZZI
      # 00020010            16.  OPRTAB  EQU      $20100               ;TABELLA DI OPERANDI
      # 0000000A            17.  TLENGTH EQU      10                   ;LUNGHEZZA DELLA TABELLA
      # 00020500            18.  BYTETAB EQU      $20500               ;TABELLA DI BYTE
                            19.  *
00010000  720A              20.           MOVE.L  #TLENGTH,D1          ;PONE IL CONTATORE A 10
00010002  207C 00020000     21.           MOVEA.L #ADDRTAB,A0          ;TABELLA DI INDIRIZZI
00010008  227C 00020500     22.           MOVEA.L #BYTETAB,A1          ;TABELLA DI BYTE
0001000E  4280              23.           CLR.L   D0                   ;INDICE PER GLI OPERANDI
                            24.  *
00010010  12F0 0D26 0008    25.  LOOP     MOVE.B  ([8,A0],D0.L*4,1),(A1)+ ;TRASFERISCE BYTE SUCC.
          0001
00010018  0680 00000001     26.           ADDI.L  #1,D0                ;INCREMENTA L'INDICE
0001001E  0481 00000001     27.           SUBI.L  #1,D1                ;DECREMENTA IL CONTATORE
00010024  66 EA             28.           BNE     LOOP                 ;CONTINUA FINCHE'
                            29.  *                                     ;  CONTO = 0.
00010026  4E4F              30.           TRAP    #15                  ;RITORNA AL MONITOR
00010028  0063              31.           DC.W    $0063
                            32.  *
                            33.  *        PREDISPONE L'AREA DATI
                            34.  *
00020000                    35.           ORG     ADDRTAB
00020000  <8>               36.           DS.B    8                    ;INTESTAZIONE
00020008  00020100          37.           DC.L    OPRTAB               ;INDIRIZZO
                            38.  *
00020100                    39.           ORG     OPRTAB
00020100  <28>              40.           DS.L    10                   ;OPERANDI
                            41.  *
00020500                    42.           ORG     BYTETAB
00020500  <A>               43.           DS.B    10                   ;BYTE
                            44.  *
0002050A                    45.           END
```

Fig. 5.22 *Esempio di indirizzamento indiretto di memoria postindicizzato.*

Indirizzamento indiretto di memoria preindicizzato. Come mostrato nella Fig. 5.23, la forma preindicizzata dell'indirizzamento indiretto di memoria calcola l'indirizzo effettivo di un operando come segue:

EA = [<bd> + An + (Xn).[S]*SCALA] + <od>

dove, come prima, [S] = W o L; SCALA = 1 (nessuno scalamento), 2, 4 o 8 e <bd> e <od> sono spostamenti di 16 bit (estesi di segno) o di 32 bit. Questa modalità ha lo scopo di consentire il reperimento di un indirizzo in una tabella di indirizzi nella memoria, mediante indicizzazione e somma di uno spostamento ad An. Per tale motivo, l'operando è contenuto all'indirizzo indiretto più il valore di <od>. Se non sono specificati un registro d'indirizzo ed uno spostamento esteriore in alcuna delle modalità d'indirizzamento indiretto della memoria, allora i modi postindicizzato e preindicizzato sono identici.

Per esempio, l'istruzione

Generazione: EA = (bd + An) + Xn.DIM * SCALA + od
Sintassi di assemblatore: ([bd,An],Xn.DIM * SCALA,od)
Modo: 110

Registro d'indirizzo: An → 31 Indirizzo di memoria 0
Spostamento di base: 31 Valore esteso di segno 0
Registro indice: 31 Valore esteso di segno 0
Scala: Valore di scala
31 Indirizzo di memoria indiretto 0
Punta a
31 Valore all'indirizzo di memoria indiretto 0
Spostamento esteriore: 31 Valore esteso di segno 0
Indirizzo effettivo: 31 Operando 0

Fig. 5.23 *Indirizzamento indiretto di memoria preindicizzato.*

```
MOVE.L      ([$900,A3,D2.W*4],$100),D1
```

con (A3) = $2000 e (D2)[15:0] = $20, specifica dapprima l'indirizzo indiretto dell'operando di sorgente come il valore esadecimale

00020000	inizio della tabella di indirizzi
0900	spostamento
0080	indice * 4
00020980	

dopodiché l'offset $100 viene aggiunto al contenuto della locazione $20980 per individuare l'indirizzo di longword dell'operando. Se ($20980) = $1F000, l'operando è alla locazione di longword $1F100. Ovviamente, la CPU non interpreta il significato delle varie componenti dell'indirizzo, ma calcola semplicemente l'indirizzo dell'operando nella maniera descritta. Dopo che l'istruzione è stata eseguita, si avrà (D1) = ($0001F100) nel risultato di 32 bit.

Esempio 5-13

La Fig. 5.24 mostra la struttura di dati nella memoria per una tabella d'indirizzi che inizia alla locazione $20100. Ogni entrata nella tabella punta ad un'altra tabella, che potrebbe contenere informazioni in merito ai dispositivi di I/O nel sistema. Lo stato di ciascuno dei quattro dispositivi è contenuto in una word di 16 bit; l'offset è di una longword dall'inizio di ciascuna tabella di dispositivi. Il programma della Fig. 5.25 trasferisce lo stato di ciascuna tabella di dispositivi, ponendo tale informazione nello stack di sistema.

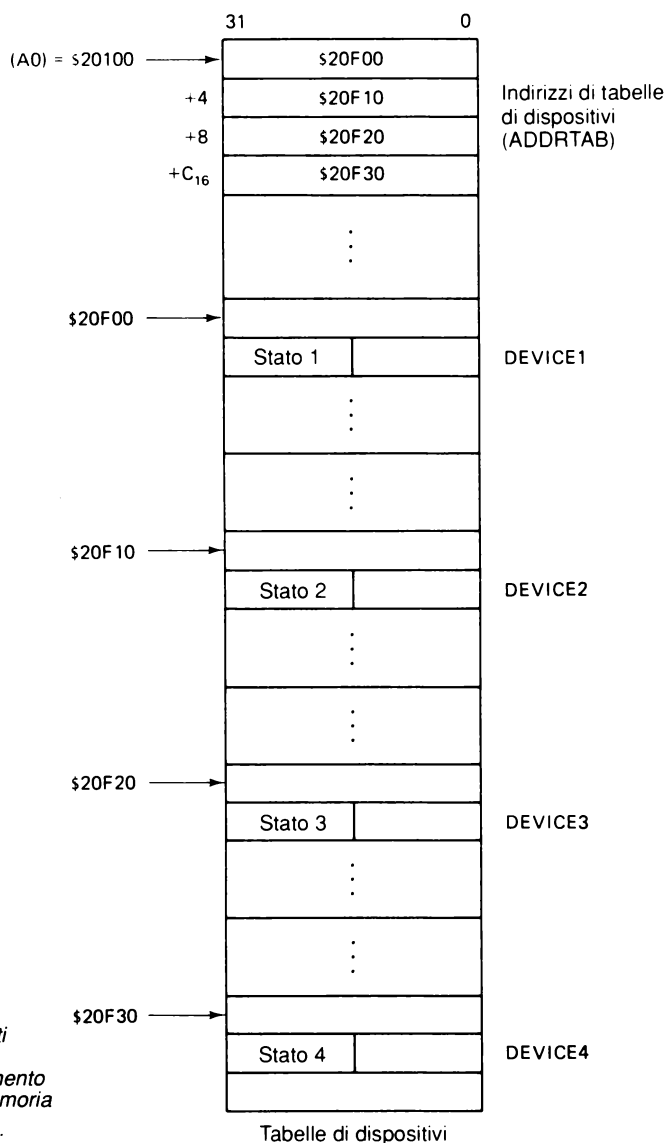



Fig. 5.24
Struttura di dati nella memoria per l'indirizzamento indiretto di memoria preindicizzato.

```
                                1.          TTL      FIGURA 5.25
                                2.          LLEN     100
                                3.  *
  = 00020F00                    4.  DEVICE1 EQU      $20F00              ;DEFINISCE GLI INDIRIZZI
  = 00020F10                    5.  DEVICE2 EQU      $20F10
  = 00020F20                    6.  DEVICE3 EQU      $20F20
  = 00020F30                    7.  DEVICE4 EQU      $20F30
                                8.  *
  = 00020100                    9.  ADDRTAB EQU      $20100              ;TAB. INDIRIZZI DI DISP.
  = 00000004                   10.  DEVCNT  EQU      4                   ;NUMERO DI DISPOSITIVI
  = 00000004                   11.  STATUS  EQU      4                   ;OFFSET A STATO
                               12.  *
                               13.  *       OUTPUT:  STATO DEL DISPOSITIVO SU STACK DI SISTEMA
                               14.  *                PER DEVCNT DISPOSITIVI
                               15.  *
00010000                       16.          ORG      $10000
00010000  4280                 17.          CLR.L    D0                  ;CONTATORE DI LOOP
00010002  207C 00020100        18.          MOVEA.L  #ADDRTAB,A0         ;INDIRIZZO BASE IN A0
                               19.  *
00010008  3F30 0D12 0004       20.  LOOP    MOVE.W   ([0,A0,D0.L*4],STATUS),-(A7)
0001000E  0680 00000001        21.          ADDI.L   #1,D0               ;DISPOSITIVO SUCCESSIVO
00010014  0C80 00000004        22.          CMPI.L   #DEVCNT,D0          ;DEVCNT CICLI
0001001A  66 EC                23.          BNE      LOOP
                               24.  *
0001001C  4E4F                 25.          TRAP     #15                 ;RITORNA AL MONITOR
0001001E  0063                 26.          DC.W     $0063
                               27.  *
00020100                       28.          ORG      ADDRTAB
00020100  00020F00             29.          DC.L     DEVICE1,DEVICE2,DEVICE3,DEVICE4
          00020F10
          00020F20
          00020F30
                               30.  *
00020110                       31.          END
```

Fig. 5.25 *Esempio d'indirizzamento indiretto di memoria preindicizzato.*

L'indirizzo iniziale di ciascuna tabella di dispositivi viene stabilito col modo preindicizzato di indirizzamento indiretto, usando A0 come indirizzo base e D0 come registro indice. Un valore "STATUS" rappresenta l'offset (spostamento esteriore) di +4 byte da questo indirizzo iniziale. Il ciclo viene eseguito quattro volte per trasferire sullo stack i valori dello stato di 16 bit. Da quel punto in avanti, un altro programma (non mostrato) elaborerà ciascun valore per determinare lo stato del dispositivo corrispondente. Tipicamente, un dispositivo può essere "occupato" o "pronto" per trasferire i dati. Un'istruzione quale

```
MOVE.W    (A7)+,D1
```

potrebbe essere usata per trasferire lo stato dallo stack ad un registro di dati per ulteriore elaborazione.

Ciascun indirizzo per le tabelle di dispositivi ed il numero di dispositivi (DEVCNT) sono fissati nel programma mostrato. Un altro programma deve provvedere a memorizzare nelle quattro tabelle le informazioni riguardanti i dispositivi, prima dell'esecuzione del programma in Fig. 5.25.

5.3.5 Indirizzamento relativo e indiretto di memoria con PC

Come descritto nel cap. 4, il modo d'indirizzamento relativo fa sì che un indirizzo venga calcolato mediante la formula generale:

Indirizzo effettivo = (PC) + <offset>

dove l'offset potrebbe essere uno spostamento intero o il valore in un registro indice, o la somma di entrambi. Quindi l'indirizzo è specificato relativamente al valore nel contatore di programma (PC). Poiché questo valore è, nella maggior parte dei casi, l'indirizzo della locazione dell'istruzione successiva da eseguire, l'offset rappresenta la distanza (in byte) tra la locazione dell'istruzione ed il valore indirizzato. L'indirizzo assoluto del valore nella memoria non è importante e in effetti qualsiasi riferimento ad esso è indipendente dalla posizione.

L'MC68020 ha cinque modalità d'indirizzamento che consentono al programmatore di descrivere un indirizzo come un offset dal contenuto del contatore di programma. Tali modalità sono formalmente identiche a quelle che impiegano un registro d'indirizzo, tranne che è usato (PC) invece di An come indirizzo di base. In alcuni casi, come nelle istruzioni di salto, l'assemblatore assegna automaticamente un modo d'indirizzamento relativo al PC. Tipicamente, la destinazione è un'etichetta nel programma, la cui locazione è specificata dall'assemblatore. L'indirizzo è calcolato come un valore di offset dal (PC) in fase di esecuzione del programma. Questo modo fondamentale d'indirizzamento relativo al PC è definito *indiretto di PC con spostamento*. La CPU dispone anche di due modi di PC con indicizzazione e di due modi d'indirizzamento indiretto di memoria con PC. Una limitazione importante dell'uso di queste modalità è che una locazione alterabile (scrivibile) non può essere indirizzata da alcun modo d'indirizzamento relativo al PC. Le modalità d'indirizzamento relativo saranno descritte nel cap. 9, allorché sarà presentata la codifica indipendente dalla posizione.

5.3.6 Indirizzamento immediato e implicito

La modalità d'indirizzamento *immediato* dell'MC68020 consente di usare valori di byte, word o longword come costanti in un'istruzione. I valori lunghi un byte o una word richiedono una word di estensione, mentre il valore lungo aggiunge due word all'istruzione nella memoria. L'assemblatore riconosce dal simbolo "#" la modalità d'indirizzamento immediato in istruzioni come:

```
MOVE.W    #50,D2
```

che trasferisce il valore 50 (decimale) nella word meno significativa di D2. Come si è definito in precedenza, i valori esadecimali sono contraddistinti dal prefisso "$", mentre i caratteri ASCII sono racchiusi tra apostrofi.

Alcune istruzioni dell'MC68020 non richiedono di specificare alcun operando, mentre altre istruzioni riguardano i registri del processore, anche se essi non vengono riferiti esplicitamente. Questi riferimenti *impliciti* possono riguardare il contatore di programma, il registro di stato, o il puntatore di stack. Per esempio, l'istruzione

```
RTS
```

che causa il ritorno da una subroutine al programma chiamante, usa il puntatore di stack per reperire l'indirizzo di ritorno dllo stack. Questa istruzione non fa riferimento esplicito al puntatore di stack né al contatore di programma, ma li modifica entrambi durante la sua esecuzione. Questi tipi di istruzioni hanno un modo d'indirizzamento che è implicito.

5.3.7 Categorie d'indirizzamento e formato della word di estensione

Le molte categorie d'indirizzamento dell'MC68020 possono essere raggruppate in categorie che determinano il tipo di accesso che può esser fatto ad un operando. Queste categorie definiscono ulteriormente i vincoli imposti ad alcune modalità d'indirizzamento. Ciascun modo d'indirizzamento unico ha un corrispondente formato di linguaggio-macchina nella memoria. Questo formato definisce il numero di word di estensione eventualmente richieste da un'istruzione per indirizzare un operando.

Categorie di indirizzamento. Le singole modalità d'indirizzamento dell'MC68020 si possono ulteriormente classificare in quattro gruppi di *categorie d'indirizzamento.* Queste categorie si riferiscono alle caratteristiche dell'operando da indirizzare. Come definito nella Tab. 5.9, tali categorie sono definite come: *dati*, *memoria*, *controllo* e *alterabile*. La categoria di dati comprende tutte le modalità tranne quella d'indirizzamento diretto di registro d'indirizzo. Ciò è logico, poiché in questo contesto si presume che l'operando in un registro d'indirizzo sia effettivamente un indirizzo e non un elemento di dati. Escluso l'indirizzamento diretto di registro, tutte le altre modalità d'indirizzamento possono far riferimento ad un operando nella memoria. Tuttavia, tutti i riferimenti alla memoria non sono considerati alterabili. Un valore immediato è ovviamente non alterabile. Inoltre, i modi d'indirizzamento relativo non fanno riferimento ad un operando alterabile. Ciò significa che i modi d'indirizzamento relativo non possono essere usati per definire indirizzi di destinazione che devono essere scritti (e quindi alterati). Per esempio, un'istruzione MOVE richiede una destinazione alterabile, che esclude le modalità immediate e relative, secondo la Tab. 5.9.

Tab. 5.9 *Categorie di modalità d'indirizzamento.*

Modalità d'indirizzamento	Modo	Registro	Dati	Memoria	Controllo	Alterabile	Sintassi dell'assemblatore
Diretto di registro di dati	000	N. reg.	x	—	—	x	Dn
Diretto di registro d'indirizzo	001	N. reg.	—	—	—	x	An
Indiretto di registro d'indirizzo	010	N. reg.	x	x	x	x	(An)
Indiretto di registro d'indirizzo con postincremento	011	N. reg.	x	x	—	x	(An)+
Indiretto di registro d'indirizzo con predecremento	100	N. reg.	x	x	—	x	–(An)
Indiretto di registro d'indirizzo con spostamento	101	N. reg.	x	x	x	x	$(d_{16}$,An)
Indiretto di registro d'indirizzo con indice (spostamento di 8 bit)	110	N. reg.	x	x	x	x	$(d_8$,An,Xn)
Indiretto di registro d'indirizzo con indice (spostamento di base)	110	N. reg.	x	x	x	x	(bd,An,Xn)
Indiretto di memoria postindicizzato	110	N. reg.	x	x	x	x	([bd,An],Xn,od)
Indiretto di memoria preindicizzato	110	N. reg.	x	x	x	x	([bd,An],Xn,od)
Assoluto corto	111	000	x	x	x	x	(xxx).W
Assoluto lungo	111	001	x	x	x	x	(xxx).W
Indiretto di contatore di programma con spostamento	111	010	x	x	x	—	$(d_{16}$,PC)
Indiretto di contatore di programma con indice (spostamento di 8 bit)	111	011	x	x	x	—	$(d_8$,PC,Xn)
Indiretto di contatore di programma con indice (spostamento di base)	111	011	x	x	x	—	(bd, PC,Xn)
Indiretto di memoria di PC postindicizzato	111	011	x	x	x	—	([bd,PC],Xn],od)
Indiretto di memoria di PC preindicizzato	111	011	x	x	x	—	([bd,PC],Xn],od)
Immediato	111	100	x	x	—	—	#<dato>

Nota: I modi d'indirizzo effettivo possono essere classificati per la maniera in cui possono essere usati. Le seguenti classificazioni saranno impiegate nelle definizioni delle istruzioni.

Dati — Se un modo d'indirizzo effettivo può essere usato per fare riferimento ad operandi di dati, allora esso è considerato un modo d'indirizzo effettivo d'indirizzamento di dati.

Memoria — Se un modo d'indirizzo effettivo può essere usato per fare riferimento ad operandi di memoria, allora esso è considerato un modo d'indirizzo effettivo d'indirizzamento della memoria.

Alterabile — Se un modo d'indirizzo effettivo può essere usato per fare riferimento ad operandi alterabili (scrivibili), allora esso è considerato un modo d'indirizzo effettivo d'indirizzamento alterabile.

Controllo — Se un modo d'indirizzo effettivo può essere usato per fare riferimento ad operandi di cui non è specificata la dimensione, allora esso è considerato un modo d'indirizzo effettivo d'indirizzamento di controllo.

Fonte: Motorola, Inc.

Il programmatore impiega le informazioni delle categorie per determinare quali modi d'indirizzamento sono ammessi per una particolare istruzione. Per esempio, in base alle descrizioni reperibili nello *User's Manual* della Motorola o nell'app. D, l'istruzione CLR ammette soltanto modalità d'indirizzamento alterabile di dati per l'operando di destinazione. Ciò esclude i registri d'indirizzo e gli indirizzi relativi come destinazioni, in base alla Tab. 5.9. Pertanto, l'istruzione

```
CLR        A1
```

non è ammessa. Nelle discussioni delle istruzioni nei prossimi capitoli, l'indirizzamento ammesso sarà descritto in termini di queste categorie.

Si dovrebbero effettuare alcune considerazioni in merito alle categorie nella Tab. 5.9:

(a) Le istruzioni che richiedono un indirizzamento di dati- alterabile non possono impiegare registri d'indirizzo come operandi.

(b) Le modalità d'indirizzamento relativo fanno riferimento ad operandi che non sono alterabili (scrivibili) da istruzioni dell'MC68020.

(c) L'indirizzamento indiretto di registro è ammesso per tutte le istruzioni che richiedono operandi. Il medesimo commento vale per le altre modalità che sono considerate comprensive di tutte e quattro le categorie.

Formato della word di estensione. I formati per le word di estensione richiesti dalle varie modalità d'indirizzamento sono mostrati nella Fig. 5.26. Per un'istruzione a singolo indirizzo, come la seguente:

```
CLR.[S]    <indirizzo>
```

dove [S] = B, W o L, l'indirizzo può essere specificato in un modo qualsiasi. Se non c'è alcuna estensione all'indirizzo, l'istruzione occupa una locazione di word di 16 bit nella memoria. Un indirizzo assoluto richiede una sola word (16 bit) o due word addizionali (32 bit) per specificare l'indirizzo. Nelle modalità con indicizzazione e spostamento, sono richieste una o più word di estensione. Quando s'impiega l'indirizzamento indiretto di memoria, ciascuno spostamento richiede zero, una o due word di estensione. Quindi l'istruzione CLR, con base e spostamento esteriore di 32 bit, che impiega l'indirizzamento indiretto di memoria, occuperebbe in totale sei word nella memoria: una word per l'istruzione e cinque word per l'estensione, con due spostamenti di 32 bit. Un'istruzione a doppio indirizzo come MOVE potrebbe richiedere fino a cinque word di estensione per ogni indirizzo, per un totale di 11 word.

Formato d'istruzione con singolo indirizzo effettivo

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	X	X	Indirizzo effettivo					
										Modo			Registro		

MC68020, word di estensione di formato breve

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D/A	Registro			W/L	Scala		0	Spostamento							

MC68020, word di estensione di formato completo

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D/A	Registro			W/L	Scala		1	BS	IS	DIM BD		0	I/IS		
Spostamento di base (0, 1 o 2 word)															
Spostamento esteriore (0, 1 o 2 word)															

Campo	Definizione
Registro	Numero di registro indice
D/A	Tipo di registro indice: 0 = Dn 1 = An
W/L	Dimensione dell'indice di word/longword: 0 = word estesa di segno 1 = longword
Scala	Fattore di scala: 00 = 1 01 = 2 10 = 4 11 = 8
BS	Soppressione della base: 0 = registro di base aggiunto 1 = registro di base soppresso
IS	Soppressione dell'indice: 0 = valutazione e aggiunta dell'operando di indice 1 = soppressione dell'operando di indice
DIM BD	Dimensione dello spostamento di base: 00 = riservato 01 = spostamento nullo 10 = spostamento di word 11 = spostamento di longword
I/IS	Selezione indice/spostamento: Operando indiretto e di indicizzazione, determinato insieme col bit 6, soppressione dell'indice.

Fig. 5.26 *Formati della word di estensione.*

IS	Indice/ indiretto	Operazione
0	000	Nessun indirizzamento indiretto di memoria
0	001	Indiretto preindicizzato con spostamento nullo
0	010	Indiretto preindicizzato con spostamento di word
0	011	Indiretto preindicizzato con spostamento di longword
0	100	Riservato
0	101	Indiretto postindicizzato con spostamento nullo
0	110	Indiretto postindicizzato con spostamento di word
0	111	Indiretto postindicizzato con spostamento di longword
1	000	Nessun indirizzamento indiretto di memoria
1	001	Indiretto di memoria con spostamento nullo
1	010	Indiretto di memoria con spostamento di word
1	011	Indiretto di memoria con spostamento di longword
1	100-111	Riservato

Modo d'indirizzamento	Modo	Registro
Diretto di registro di dati	000	num. reg.
Diretto di registro d'indirizzo	001	num. reg.
Indiretto di registro d'indirizzo	010	num. reg.
Indiretto di registro d'indirizzo con postincremento	011	num. reg.
Indiretto di registro d'indirizzo con predecremento	100	num. reg.
Indiretto di registro d'indirizzo con spostamento	101	num. reg.
Indiretto di registro d'indirizzo e di memoria con indice	110	num. reg.
Assoluto corto	111	000
Assoluto lungo	111	001
Indiretto di contatore di programma con spostamento	111	010
Indiretto di contatore di programma e di memoria con indice	111	011
Dato immediato	111	100
Riservato per uso futuro dalla Motorola	111	101
Riservato per uso futuro dalla Motorola	111	110
Riservato per uso futuro dalla Motorola	111	111

Fig. 5.26 *(continuazione)*

Esempio 5-14

Il listato nella Fig. 5.27 indica alcuni possibili formati di word di estensione per le istruzioni. Le istruzioni in linguaggio-macchina possono essere decodificate facendo riferimento alla Fig. 5.26.

```
                            1.           TTL      FIGURA 5.27
00010000                    2.           ORG      $10000
                            3.   *
                            4.   *       MODI D'INDIRIZZAMENTO E WORD DI ESTENSIONE
                            5.   *
                            6.   *                SINGOLO INDIRIZZO
                            7.   *
00010000  4201              8.           CLR.B    D1
00010002  4281              9.           CLR.L    D1
00010004  4259             10.           CLR.W    (A1)+
00010006  4278 1000        11.           CLR.W    $1000
0001000A  4269 1000        12.           CLR.W    $1000(A1)
0001000E  4271 2010        13.           CLR.W    $10,(A1,D2)
                           14.   *
00010012  4271 1210        15.           CLR.W    ($10,A1,D1.W*2)
00010016  4271 A122 0002   16.           CLR.W    ([2,A1,A2.W],4)
          0004
0001001E  4271 AD22 1000   17.           CLR.W    ([$1000,A1,A2.L*4],$2000)
          2000
00010026  4271 AF33        18.           CLR.W    ([$1FFF0,A1,A2.L*8],$2FFF0)
          0001FFF0
          0002FFF0
                           19.   *
                           20.   *                DOPPIO INDIRIZZO
                           21.   *
00010032  1200             22.           MOVE.B   D0,D1
00010034  2200             23.           MOVE.L   D0,D1
00010036  3519             24.           MOVE.W   (A1)+,-(A2)
00010038  31F8 0020 0040   25.           MOVE.W   $20,$40
0001003E  31F8 1000 2000   26.           MOVE.W   $1000,$2000
00010044  3229 0010        27.           MOVE.W   $10(A1),D1
00010048  3569 0010 0010   28.           MOVE.W   $10(A1),$10(A2)
0001004E  3569 1000 2000   29.           MOVE.W   $1000(A1),$2000(A2)
00010054  35B1 1210 2E00   30.           MOVE.W   ($10,A1,D1.W*2),(A2,D2.L*8)
0001005A  33B1 A122 0002   31.           MOVE.W   ([2,A1,A2.W],4),(A1,D2)
          0004 2000
00010064  37B1 AF33        32.           MOVE.W   ([$1FFF0,A1,A2.L*8],$2FFF0),([$1000,A3,D2.L])
          0001FFF0
          0002FFF0  2921
          1000
00010074  37B1 AF33        33.           MOVE.W   ([$1FFF0,A1,A2.L*8],$2F000),([$3FFF0,A3,A4.L*8],$4FFF0)
          0001FFF0
          0002F000
          0003FFF0  CF33
          0004FFF0
                           34.
```

Fig. 5.27 *Esempio di formati di word di esteneione.*

ESERCIZI

5.3.1 Si elenchino alcuni motivi per cui gli operandi indirizzati relativamente al contatore di programma possono essere letti e manipolati ma le loro locazioni non possono essere scritte.

5.3.2 L'MC68020 non ammette che l'istruzione CLR specifichi un registro d'indirizzo come destinazione. Questa restrizione sull'impiego di CLR con registri d'indirizzo potrebbe essere un problema? Come può un programma indirizzare la locazione 0?

5.3.3 Si discutano le varie modalità d'indirizzamento in termini dei valori da definire prima dell'assemblaggio e di quelli che possono essere definiti o modificati durante l'esecuzione del programma. (Si supponga che il programma non modificherà le proprie istruzioni nella memoria.)

5.3.4 Si discutano i vantaggi e gli svantaggi del modo d'indirizzamento assoluto corto rispetto al modo d'indirizzamento assoluto lungo.

5.3.5 Se (A1) = $1000, si determini l'indirizzo dell'operando per le seguenti istruzioni:

```
(a) CLR.B     $FFFF (A1)
(b) MOVE.B    (A1)+,D1
(c) MOVE.W    -(A1),D1
```

5.3.6 Che cosa fanno le seguenti istruzioni?

```
(a) MOVE.L    4 (A0),(A0)
(b) MOVE.W    $9000,D1
(c) MOVE.B    0(A0,D1),D1
```

5.3.7 Si confronti la modalità d'indirizzamento assoluto corto col modo di spostamento indiretto, quando il valore dello spostamento è $8000.

5.3.8 Si consideri l'istruzione

```
MOVE.L    (0,A1,D1,W*4),(A2)+
```

in un ciclo in cui (D1) viene incrementato ad ogni iterazione. Inizialmente (A1) = $00010000 e (D1) = 0.

(a) Se (D1) viene incrementato di 1 ogni volta, si dica come cambia l'indirizzo dell'operando ad ogni iterazione del ciclo.
(b) Se (D1) viene incrementato di 16 ogni volta, che tipo di struttura di dati è oggetto dell'indirizzamento?

5.3.9 Si supponga che:

```
(A1) = (A2)    = $00010000
(D1)[15:0]     = $0010
($10010)       = $20000
($20010)       = $30000
($20016)       = $100
$(30006)       = $200
```

Qual è il risultato nella destinazione per le seguenti istruzioni?

```
(a) MOVE.W    ([$10,A1],D1.W*1, 6), D3
(b) MOVE.W    ([$10,A1, A2.W*1], 6), D4
```

5.3.10 Si modifichi il programma descritto nell'esempio 5.10 per determinare la lunghezza della lista nella memoria (specificata nella prima word dell'intestazione come numero esadecimale di word nella lista). Si esegua un test su ciascun valore della lista nella memoria per verificare se è nullo e si utilizzi D4 come un contatore di valori nulli.

5.3.11 Si modifichi il programma dell'esempio 5.12 per salvare l'indirizzo di qualsiasi longword che contenga un valore zero in OPRTAB dopo che il byte è stato trasferito alla tabella di valori lunghi un byte. La nuova tabella di indirizzi di 32 bit dovrebbe essere memorizzata a partire dalla locazione $30000. L'ultimo elemento inserito dopo tutti gli indirizzi sia $FFFFFFFF per indicare la fine della tabella.

5.3.12

Si modifichi il programma dell'esempio 5.13 per esaminare i valori di stato posti in ordine sullo stack. Se lo stato è zero, si trasferisca l'indirizzo della tabella di dispositivi ad una tabella di quattro indirizzi, indirizzata da A3. Se lo stato è diverso da zero, si scriva $FFFFFFFF all'indirizzo che corrisponde al dispositivo in esame. Quindi la tabella finale conterrà l'indirizzo della tabella di dispositivi oppure –1 per ciascuno dei quattro dispositivi nell'ordine.

5.4 RIEPILOGO DELLE MODALITA' D'INDIRIZZAMENTO

L'MC68020 ha un certo numero di modalità d'indirizzamento per fare riferimento ad un operando in un registro, nell'istruzione stessa o nella memoria. Ogni istruzione dell'MC68020 ammette certe modalità d'indirizzamento per definire la locazione di eventuali operandi (se esistono). Con alcune limitazioni, le modalità d'indirizzamento possono essere combinate con qualsiasi operazione per formare un'istruzione completa. Queste limitazioni saranno trattate in dettaglio quando s'incontreranno istruzioni particolari nei prossimi capitoli. In questo paragrafo ci si concentrerà sull'impiego di vari modi d'indirizzamento e dei loro formati in linguaggio assembler.

```
                           1.        TTL     FIGURA 5.28
00010000                   2.        ORG     $10000
                           3.  *
                           4.  *     ISTRUZIONE                         NOME
                           5.  *
00010000  D481             6.        ADD.L   D1,D2                      DIRETTO DI REGISTRO
00010002  D489             7.        ADD.L   A1,D2
                           8.  *
00010004  D47C 0064        9.        ADD.W   #100,D2                    IMMEDIATO
                          10.  *
00010008  D479 0001F000   11.        ADD.W   $1F000,D2                  ASSOLUTO
0001000E  D379 0002F000   12.        ADD.W   D1,$2F000
                          13.  *
00010014  D451            14.        ADD.W   (A1),D2                    INDIRETTO
                          15.  *
00010016  D461            16.        ADD.W   -(A1),D2                   PREDECREMENTO; POSTINCREMENTO
00010018  D359            17.        ADD.W   D1,(A1)+
                          18.  *
0001001A  D469 1000       19.        ADD.W   $1000(A1),D2               INDIRETTO CON SPOSTAMENTO
0001001E  D471 0970       20.        ADD.W   ($1F000,A1),D2
          0001F000
00010026  D479 0001F000   21.        ADD.W   ($1F000),D2                  ;SOPPRIME A1
                          22.  *
0001002C  D471 1820       23.        ADD.W   $20(A1,D1.L),D2            INDIRETTO CON INDICE
00010030  D471 1A20       24.        ADD.W   ($20,A1,D1.L*2),D2
00010034  D471 1B20 1F00  25.        ADD.W   ($1F00,A1,D1.L*2),D2
0001003A  D471 1D30       26.        ADD.W   ($1F000,A1,D1.L*4),D2
          0001F000
00010042  D470 1DB0       27.        ADD.W   ($1F000,D1.L*4),D2           ;SOPPRIME A2
          0001F000
0001004A  D470 1990       28.        ADD.W   (D1.L),D2                    ;INDIRETTO DI REGISTRO DATI
                          29.  *
0001004E                  30.        END
```

Fig. 5.28 *Modalità d'indirizzamento di registro.*

```
                              1.      TTL     FIGURA 5.29
00010000                      2.      ORG     $10000
                              3.  *
                              4.  *   ISTRUZIONE                                  NOME
                              5.  *
00010000  D471 1926 0020      6.      ADD.W   ([$20,A1],D1.L,$10),D2              POST-INDICIZZATO
          0010
00010008  D471 1B26 1F00      7.      ADD.W   ([$1F00,A1]D1.L*2,$1000),D2
          1000
00010010  D471 1B37           8.      ADD.W   ([$1F000,A1],D1.L*2,$1F000),D2
          0001F000
          0001F000
0001001C  D471 0962 1F00      9.      ADD.W   ([$1F00,A1],$1000),D2               ;SOPPRIME INDICE
          1000
00010024  D471 0961 1F00     10.      ADD.W   ([$1F00],A1]),D2                    ;SOPPRIME SPOSTAMENTO ESTERNO
0001002A  D471 0951          11.      ADD.W   ([A1]),D2                           ;SOPPRIME SPOSTAMENTO DI BASE
0001002E  D470 19A5 1F00     12.      ADD.W   ([$1F00],D1.L),D2                   ;SOPPRIME A1
00010034  D470 09E1 1F00     13.      ADD.W   ([$1F00]),D2                        ;INDIRETTO ASSOLUTO
0001003A  D470 09D1          14.      ADD.W   ([]),D2                             ;LOCAZIONE 0
                             15.
0001003E  D471 1922 0020     16.      ADD.W   ([$20,A1,D1.L],$10),D2              PRE-INDICIZZATO
          0010
00010046  D471 1B22 1F00     17.      ADD.W   ([$1F00,A1,D1.L*2],$1000),D2
          1000
0001004E  D471 1B33          18.      ADD.W   ([$1F000,A1,D1.L*2],$1F000),D2
          0001F000
          0001F000
0001005A  D470 1BA3 1F00     19.      ADD.W   ([$1F00,D1.L*2],$1F000),D2          ;SOPPRIME A1
          0001F000
00010064  D470 1933          20.      ADD.W   ([D1.L]$1F000),D2                   ;SOPPRIME SPOST. BASE
          0001F000
0001006C  D470 09F3          21.      ADD.W   ([$1F000],$1F000),D2                ;SOPPRIME A1 E D1
          0001F000
          0001F000
00010078  D470 09D3          22.      ADD.W   ([],$1F000),D2                      ;ASSOLUTO
          0001F000
                             23.  *
00010080                     24.      END
```

Fig. 5.29 *Modalità d'indirizzamento indiretto di memoria.*

La Tab. 5.10 riassume in breve i modi d'indirizzamento ed elenca i possibili impieghi di una particolare modalità. Alcuni esempi delle modalità sono mostrati nelle Figg. 5.28 e 5.29. Nei modi diretti di registro, l'operando è contenuto in un registro d'indirizzo o in un registro di dati. Per il modo immediato, altresì noto come modo "letterale", l'operando è definito nella memoria come parte dell'istruzione in linguaggio-macchina. Queste modalità fondamentali sono sufficienti per molti programmi in cui gli operandi non fanno parte di una struttura di dati nella memoria, ma sono trattati singolarmente, come nei calcoli matematici o nei trasferimenti di dati. Le modalità d'indirizzamento assoluto sono impiegate per far riferimento a indirizzi prefissati nella memoria: di solito in relazione all'hardware, come per i dispositivi di I/O. Quando la struttura di dati nella memoria è più complessa, un modo d'indirizzamento indiretto è di solito più conveniente per indirizzare un operando. Tali strutture di dati saranno trattate nel cap. 9.

Tab.5.10 Impiego dei modi di indirizzamento.

Modo d'indirizzamento	Esempio d'impiego
Diretto di registro	Operazioni matematiche o di trasferimento di dati su operandi contenuti in registri
Immediato	Incremento o inizializzazione di valori in registri o nella memoria
Assoluto	Riferimento a locazioni riguardanti l'hardware
Indiretto	Operazioni su valori di dati nella memoria
Predecremento o postincremento	Operazioni su valori in array, stack e code
Indiretto con spostamento	Riferimento ad un operando nella memoria da un indirizzo di base
Indiretto con indice	Riferimento ad un operando in una tabella di valori nella memoria
Indiretto di memoria	Riferimento ad una tabella di indirizzi nella memoria
Indiretto di memoria postindicizzato	Indice in una tabella di operandi nella memoria individuata da un indirizzo indiretto
Indiretto di memoria preindicizzato	Indice in una tabella di indirizzi nella memoria per determinare un indirizzo indiretto
Relativo al PC	Creazione di riferimenti indipendenti dalla posizione ad operandi o indirizzi

ESERCIZI

5.4.1 Per ciascuna delle istruzioni che impiegano l'indirizzamento indiretto nella Fig. 5.28, si supponga che, prima che l'istruzione venga eseguita, (A1) = $00010000 e (D1) = $00010000. Qual è l'indirizzo indiretto dell'operando di sorgente?

5.4.2 Prima dell'esecuzione di ciascuna istruzione in Fig. 5.29, si supponga che:

(A1) = $00010000
(D1) = $00010000

($0000) = $10000
($1F00) = $100
($10000) = $200
($10020) = $300
($11F00) = $400
($1F000) = $500
($20020) = $600
($21F00) = $700
($2F000) = $800
($31F00) = $900
($4F000) = $A00

(a) Qual è l'indirizzo indiretto fondamentale per ciascun operando di sorgente?
(b) Qual è l'indirizzo dell'operando per ciascun operando di sorgente?

5.4.3

Si supponga che la locazione 0 contenga $01234567. Qual è il risultato dell'istruzione

```
CLR.W     ([0])
```

dopo l'esecuzione?

5.4.4

Si supponga che un programma abbia la sequenza di istruzioni riportata di seguito:

```
ADD.W     PRIMO(PC), D1
.         .
.         .
ADD.W     PRIMO(PC), D1
          .
.         .
END
```

Il codice in linguaggio-macchina per le due istruzioni ADD è il medesimo? Lo sarebbe se (PC) fosse sostituito da (A1)?

CAPITOLO 6

TRASFERIMENTO DI DATI, CONTROLLO DEL PROGRAMMA E SUBROUTINE

Vari capitoli sono dedicati ad una discussione dell'insieme di istruzioni in linguaggio assembler dell'MC68020. Lo scopo di questi capitoli è quello di analizzare in dettaglio ciascuna istruzione e d'illustrarne l'impiego nei programmi in linguaggio assembler. Tali discussioni iniziano in questo capitolo. Le istruzioni e le loro varianti sono separate in categorie in base alle operazioni che svolgono. In questo capitolo sono discusse le istruzioni elencate nella Tab. 6.1 per il *trasferimento di dati*, il *controllo del programma* e l'*impiego di subroutine*. Nei capitoli successivi saranno presentate le istruzioni per le operazioni aritmetiche, logiche e simili.

L'istruzione MOVE è l'istruzione principale nella categoria di trasferimento dei dati. Essa non ha molte restrizioni sull'ubicazione e la lunghezza degli operandi che possono essere trasferiti tra la CPU e la memoria. Nella Tab. 6.1 sono elencate anche due varianti di questa istruzione. Esse sono distinte dall'istruzione MOVE per l'aggiunta di una lettera di suffisso, Q o M, per formare MOVEQ e MOVEM. La forma "rapida" (*quick*) MOVEQ è un'istruzione di una sola word per caricare un valore di dati in un registro di dati. L'istruzione MOVEM è una variante che consente di trasferire il contenuto di un gruppo selezionato di registri da o verso locazioni di memoria consecutive. Queste due varianti possono risultare più efficienti dell'istruzione MOVE in certe circostanze.

Le istruzioni per il controllo del programma sono usate per definire il flusso di controllo entro un programma. Le istruzioni BRA e JMP causano un trasferimento incondizionato del controllo. Le istruzioni Bcc e DBcc producono un salto quando si verificano certe condizioni, definite dai codici di condizione. L'istruzione CMP e le sue varianti, anch'esse elencate nella Tab. 6.1, sono incluse nella discussione in questo capitolo perché definiscono i codici di condizione in base ai valori degli operandi. Una descrizione del registro dei codici di condizione (CCR) è stata già svolta nel par. 4.2.

Tab. 6.1 *Istruzioni selezionate.*

	Trasferimento di dati
MOVE	Trasferimento
MOVEQ	Trasferimento rapido (immediato)
MOVEM	Trasferimento di registri
EXG	Scambio di registri
SWAP	Scambio delle metà di registri di dati
	Controllo del programma
Incondizionato	
BRA	Salto senza condizioni
JMP	Salto
Salto condizionato e confronto	
Bcc	Salto condizionato
DBcc	Condizione di test, decremento e salto
CMP	Confronto
CMPI	Confronto immediato
CMPM	Confronto di memoria
TST	Test
	Subroutine
BSR	Salto relativo a subroutine
JSR	Salto a subroutine
RTR	Ritorno e ripristino (CCR)
RTS	Ritorno da subroutine

Le subroutine possono essere chiamate mediante le istruzioni BSR o JSR. L'esecuzione di un'istruzione RTR o RTS nella subroutine causa la restituzione del controllo al programma chiamante. Il salvataggio ed il ripristino dell'indirizzo di ritorno vengono gestiti automaticamente dalla CPU quando le istruzioni di chiamata e di ritorno sono eseguite nella sequenza corretta.

In questo capitolo le istruzioni sono definite in termini della sintassi dell'assemblatore, della lunghezza dell'operando, dei modi d'indirizzamento validi e dell'effetto sui codici di condizione. L'insieme di istruzioni completo è presentato nell'app. D.

6.1 TRASFERIMENTO DI DATI

Le istruzioni MOVE, MOVEQ, MOVEM, EXG, e SWAP rappresentano istruzioni di trasferimento di dati. L'istruzione MOVE è la più flessibile e, di conseguenza, la più frequentemente usata. La variante "rapida", MOVEQ, trasferisce un valore di dati di 8 bit ad un registro di dati specificato. Il suffisso "M" della variante MOVEM è l'iniziale di "multiplo": questa istruzione salva nella memoria un gruppo di registri selezionato o ne ripristina il contenuto dalla memoria. Le istruzioni EXG e SWAP trasferiscono i dati tra ed entro i registri.

La Tab. 6.2 è un sommario delle istruzioni trattate in questo paragrafo. Per ogni istruzione, sono elencate la sintassi dell'assemblatore, le lunghezze valide degli operandi, le possibili modalità d'indirizzamento e i codici di condizione interessati dall'istruzione. Quando è possibile un certo numero di modalità per l'operando di sorgente o di destinazione, le modalità d'indirizzamento sono definite in termini delle categorie d'indirizzamento presentate nel cap. 5. La categoria alterabile di dati esclude i modi d'indirizzamento diretto di registro d'indirizzo, relativo ed immediato. La categoria alterabile di controllo proibisce i modi d'indirizzamento diretto di registro, con postincremento, con predecremento, relativo ed immediato. La categoria di controllo non ammette i modi d'indirizzamento diretto di registro, con postincremento, con predecremento e immediato.

6.1.1 L'istruzione MOVE

L'istruzione MOVE effettua il trasferimento di dati tra registri, o tra registri e la memoria, o tra differenti locazioni della memoria. Il suo formato è:

```
MOVE.<l>    <EAs>,<EAd>
```

in cui <l> = B, W o L per indicare operandi di 8 bit, di 16 bit o di 32 bit, rispettivamente. Soltanto la porzione della locazione di destinazione specificata da <l> è sostituita dall'operazione

$$(EAd)[l] \leftarrow (EAs)[l]$$

dove [l] designa gli appropriati bit interessati dall'operazione. Se An è specificato come un operando di sorgente, allora la lunghezza dev'essere di una word o di una longword (W o L), poiché non sono ammesse operazioni di byte su registri d'indirizzo. La modalità d'indirizzamento di destinazione dev'essere alterabile di byte, il che esclude l'impiego di registri d'indirizzo e di indirizzi relativi. La variante MOVEA dell'istruzione trasferisce valori a registri d'indirizzo.

Tab. 6.2 *Istruzioni per il trasferimento di dati.*

Istruzione	Sintassi	Lunghezza dell'operando (bit)	Modi d'indirizzamento: Sorgente	Modi d'indirizzamento: Destinazione	Codici di condizione interessati
Trasferimento	MOVE.<l> <EAs>,<EAd>	8, 16, 32	Tutti[2]	Alterabile di dati	N, Z V = {0}, C = {0}
Trasferimento rapido	MOVEQ #<d_8>,<Dn>	32	Immediato (esteso di segno)	Dn	N, Z V = {0}, C = {0}
Trasferimento di più registri	MOVEM.<l_1> <lista>,<EA>	16 o 32	Lista di registri	Alterabile di controllo o predecremento	Nessuno
	MOVEM.<l_1> <EA>,<lista>	16 o 32	Controllo o postincremento	Lista di registri	Nessuno
Scambio di registri	EXG <Rx>,<Ry>	32	Rx	Ry	Nessuno
Scambio delle metà di un registro	SWAP <Dn>	16	(Dn)[31:16] ⟷ (Dn)[15:0]	—	N, Z V = {0}, C = {0}

Note:

1. <EA> indirizzo effettivo
   <l> B, W o L
   <l_1> W o L
   <Rn> uno qualunque di Dn o An
   <lista> lista di registri

2. Se la dimensione dell'operando è di 1 byte, <An> non può essere una sorgente.

3. Nell'istruzione MOVEM.W, gli operandi trasferiti ad un registro vengono estesi di segno a 32 bit nel registro di destinazione.

Il valore trasferito dall'istruzione MOVE è trattato come un intero con segno della lunghezza specificata. I codici di condizione N e Z sono fissati come risultato dell'operazione, mentre V e C sono posti a {0}. Quindi è possibile effettuare dei test sugli operandi per valori nulli o negativi subito dopo un'istruzione MOVE. Per esempio, la sequenza:

```
MOVE.W    LUNGH,D1
BEQ       FINE
```

causa un salto all'istruzione in FINE se il contenuto della locazione LUNGH fosse zero, poiché BEQ (*Branch if EQual zero*: salta se uguale a zero) produce un salto quando Z = {1}; altrimenti, l'esecuzione del programma continuerebbe con l'istruzione successiva. Questa sequenza potrebbe essere utilizzata, ad esempio, per trattare una tabella di dati di lunghezza assegnata. Se la lunghezza contenuta nella locazione LUNGH fosse zero, allora le istruzioni per trattare i valori dei dati sarebbero saltate e il controllo sarebbe trasferito all'istruzione etichettata FINE.

L'istruzione

```
MOVE.B    #$FF,D1
```

pone Z = {0} poiché il valore immediato non è zero, ma pone anche N = {1} poiché il valore di 8 bit è considerato negativo.

L'istruzione MOVE è usata nella maggior parte degli esempi in questo libro, per cui esempi specifici del suo impiego non sono forniti qui. Invece, nel prossimo sottoparagrafo, saranno presentate le varianti dell'istruzione MOVE, ciascuna delle quali sarà confrontata con l'istruzione di base.

6.1.2 Varianti di MOVE

MOVEQ è un'istruzione di una word che ha la forma simbolica:

```
MOVEQ     #<d8>,<Dn>
```

dove <d_8> è una costante di 8 bit. Il valore di 8 bit è esteso di segno a 32 bit e trasferito a Dn. Questa istruzione è molto efficiente quando si carica un registro di dati con una costante decimale nell'intervallo da −128 a +127. La sua esecuzione richiede soltanto tre cicli di macchina, ragion per cui è definita "rapida" (*quick*). Le istruzioni MOVE standard prelevate dalla memoria principale richiedono più cicli per tutti i trasferimenti, tranne quelli da registro a registro. I codici di condizione sono interessati dall'istruzione MOVEQ così come lo sono per l'istruzione MOVE, consentendo di effettuare un test per valori nulli o negativi.

L'istruzione per azzerare un registro:

```
MOVEQ     #0,D1
```

ha il medesimo effetto di CLR.L D1, poiché entrambe le istruzioni agiscono sull'intera lunghezza di 32 bit del registro. Non c'è alcuna differenza fondamentale nell'efficienza operativa di queste due istruzioni.

Un'altra variante dell'istruzione MOVE è MOVEM, che trasferisce dati tra registri del processore e locazioni di memoria. Tale istruzione ha la forma simbolica:

```
MOVEM.<l1>     <lista>,<EA>
```

dove i registri nella <lista> devono essere trasferiti in locazioni di memoria a partire dall'indirizzo <EA>; la word meno significativa dell'intero contenuto del registro viene trasferita se <l_1> = W o L, rispettivamente. La sintassi per <lista> è mostrata nella Tab. 6.3(a). Quindi l'istruzione:

```
MOVEM.W     D0/D1/A2-A4,$11000
```

trasferisce il contenuto meno significativo di D0, D1, A2, A3 e A4 alle locazioni $11000, $11002, S11004, $11006 e $11008, rispettivamente. Per ripristinare i valori nei registri, il formato è:

```
MOVEM.<l1>     <EA>,<lista>
```

dove <l_1> e <lista> hanno il medesimo significato di prima. Tuttavia, se operandi di lunghezza di word sono trasferiti a registri (l_1 = W), gli operandi sono estesi di segno a 32 bit nei registri.

Quando la destinazione per un'istruzione MOVEM è la memoria, sono ammesse soltanto le modalità d'indirizzamento alterabile di controllo e quella di predecremento. Ciò esclude per la destinazione <EA> l'indirizzamento diretto di registro, con postincremento e quello relativo al contatore di programma. Nelle modalità di controllo, i contenuti dei registri di dati vengono memorizzati nelle prime locazioni dell'area di memoria specificata, seguiti dai contenuti dei registri d'indirizzo, indipendentemente dall'ordine specificato nella lista di registri. La modalità di predecremento è usata per memorizzare i contenuti dei registri su uno stack che cresce verso il basso nella memoria, in cui l'ordine di memorizzazione è rovesciato; cioè, sono memorizzati per primi i registri d'indirizzo, seguiti dai registri di dati.

Un trasferimento dalla memoria ai registri ammette soltanto le modalità d'indirizzamento alterabile di controllo o di postincremento per gli operandi di sorgente. I registri vengono trasferiti nell'ordine D0, D1, ..., A0, ..., A7, indipendentemente dall'ordine nella lista. Si presume che gli indirizzi della locazioni di memorizzazione siano crescenti nella memoria. L'indirizzamento di postincremento preleva dallo stack i registri designati. La Tab. 6.3(b) riassume l'ordine di trasferimento per i vari casi.

Tab. 6.3 *L'istruzione MOVEM.*

(a) Lista di registri

(1) I registri di dati selezionati sono separati da "/"
(p. es., D1/D3/D4)

(2) I registri consecutivi sono specificati da "-"
(p. es., D1-D4)

(b) Ordine di trasferimento

Ordine	Tipo
D0-D7, A0-A7 a indirizzi superiori	1. Da registro a memoria (alterabile di controllo) 2. Da memoria a registro
A7-A0, D7-D0 a indirizzi inferiori	Da registro a memoria (predecremento)

Esempio 6-1

L'istruzione MOVEM è usata in vari modi nel programma mostrato nella Fig. 6.1. La prima MOVEM salva il contenuto di 32 bit di D4, D3, D2 e D1 sullo stack di sistema. La seconda MOVEM trasferisce quattro word dalle locazioni $11000, $11002, $1004 e $11006 ai registri D1, D2, D3 e D4, rispettivamente. I numeri sono aggiunti e la somma viene memorizzata nella locazione $11008. I registri vengono poi ripristinati dallo stack nell'ordine inverso a quello con cui erano memorizzati. Non c'è alcun test per l'overflow nel programma come possibile risultato delle addizioni di 16 bit.

6.1.3 Trasferimento di dati interno

L'istruzione EXG (*EXchanGE:* scambia) effettua lo scambio della longword di 32 bit nel registro di sorgente con quella nel registro di destinazione. I registri interessati possono essere registri di dati, registri d'indirizzo o un registro d'indirizzo ed un registro di dati. Il formato dell'istruzione è:

```
                                1.          TTL     FIGURA 6.1
                                2.          LLEN    100
00010000                        3.          ORG     $10000
                                4.  *
    = 00000063                  5.  RETURN  EQU     $0063              ;MONITOR
                                6.  *
                                7.  *       USO DELL'ISTRUZIONE MOVEM
                                8.  *
00010000  48E7 7800             9.          MOVEM.L D1-D4,-(SP)        ;SALVA I REGISTRI
                               10.                                     ; SULLO STACK
00010004  4CB9 001E            11.          MOVEM.W $11000,D1-D4       ;CARICA I DATI
          00011000
                               12.  *
0001000C  D441                 13.          ADD.W   D1,D2              ;SOMMA I NUMERI
0001000E  D642                 14.          ADD.W   D2,D3
00010010  D843                 15.          ADD.W   D3,D4
                               16.  *
00010012  33C4 00011008        17.          MOVE.W  D4,$11008          ;SALVA IL RISULTATO
                               18.  *
00010018  4CDF 001E            19.          MOVEM.L (SP)+,D1-D4        ;RIPRISTINA I REGISTRI
                               20.  *
0001001C  4E4F                 21.          TRAP    #15                ;RITORNA AL MONITOR
0001001E  0063                 22.          DC.W    RETURN
                               23.  *
                               24.
00010020                       25.          END
```

Fig. 6.1 *Uso dell'istruzione MOVEM.*

EXG <Rx>,<Ry>

in cui vengono scambiati i contenuti di Rx e Ry. Sono ammessi soltanto scambi di 32 bit; i codici di condizione restano inalterati.

L'istruzione EXG rende superflua la necessità di memorizzare temporaneamente i contenuti dei registri quando devono essere scambiati operandi di 32 bit in due registri. L'impiego dell'istruzione di scambio è illustrato nell'esempio 6.2.

L'istruzione SWAP scambia la word meno significativa con quella più significativa in un singolo registro. Questa istruzione ha la forma:

SWAP <Dn>

in cui può essere specificato soltanto un registro di dati. I codici di condizione sono impostati conformemente con l'intero risultato di 32 bit per il test di valori nulli o negativi.

Esempio 6-2

Il programma mostrato nella Fig. 6.2 illustra due metodi per scambiare i contenuti di due registri, A0 e D1. Nel primo metodo, il contenuto del registro D1 viene salvato temporaneamente nella memoria. Dopodiché, D1 può essere caricato col valore in A0, e infine A0 può essere caricato col contenuto originale di D1 prelevato dalla locazione di memoria temporanea.

Il secondo metodo impiega l'istruzione EXG per svolgere la medesima operazione con una sola istruzione.

```
                              1.          TTL      FIGURA 6.2
                              2.          LLEN     100
00010000                      3.          ORG      $10000
                              4.  *
   * 00000063                 5.  RETURN  EQU      $0063          ;MONITOR
                              6.  *
                              7.  *       USO DELL'ISTRUZIONE EXG PER SCAMBIARE D1 E A0
                              8.  *
00010000  23C1 00010014       9.          MOVE.L   D1,TEMP        ;SALVA IL CONTENUTO DI D1
00010006  2208               10.          MOVE.L   A0,D1          ;D1 := A0
00010008  2079 00010014      11.          MOVEA.L  TEMP,A0        ;A0 := D1 SALVATO
                             12. *
0001000E  C388               13.          EXG      D1,A0          ;A0 <---> D1
                             14.  *
00010010  4E4F               15.          TRAP     #15            ;RITORNA AL MONITOR
00010012  0063               16.          DC.W     RETURN
                             17.  *
00010014  <4>                18.  TEMP    DS.L     1              ;LOCAZIONE DI MEMORIA
                             19.  *                               ; TEMPORANEA
                             20.  *
00010018                     21.          END
```

Fig. 6.2 *Uso dell'istruzione EXG.*

ESERCIZI

6.1.1 Perché l'istruzione MOVE deve azzerare i codici di condizione C e V?

6.1.2 Si scriva una routine per trovare il massimo valore di un insieme di interi di 16 bit memorizzati in locazioni di memoria consecutive, confrontando i numeri nell'ordine e scambiando un valore inferiore con uno maggiore finché il registro non contiene il massimo. Si provi con e senza l'impiego dell'istruzione EXG.

6.1.3 Uno stack con N word ha il suo primo elemento nella locazione FONDO. Si scrivano le istruzioni per trasferire i valori di word in un altro stack. Si supponga che entrambi gli stack crescano verso il basso nella memoria.

6.1.4 Quale vantaggio offre l'istruzione MOVEM rispetto ad una serie di istruzioni MOVE per salvare il contenuto di registri nella memoria?

6.1.5 Si pensi ai possibili impieghi dell'istruzione SWAP. Si ricordi che il secondo byte più significativo in un registro di 32 bit non è direttamente accessibile. Quale tipo di istruzione è richiesta per accedere ad un particolare byte in un registro di 32 bit?

6.2 CONTROLLO DEL PROGRAMMA

In qualsiasi programma avanzato, è necessario selezionare la sequenza di istruzioni da eseguire in base ai risultati di calcoli, quindi il flusso di controllo seguirà percorsi differenti attraverso il programma, a seconda di tali calcoli. In alcuni casi, il trasferimento del controllo è *incondizionato*. Per esempio, l'istruzione:

```
JMP        ETICH
```

serve a trasferire incondizionatamente il controllo all'istruzione in ETICH. Altri programmi richiedono un trasferimento *condizionato* del controllo, in base ai risultati di operazioni aritmetiche o di altro tipo. Queste operazioni definiscono i codici di condizione dell'MC68020. Le istruzioni di salto condizionato esaminano questi codici di condizione per determinare se debba essere effettuato un salto oppure no.

Due istruzioni dell'MC68020, BRA e JMP, producono un trasferimento incondizionato del controllo. L'istruzione di salto BRA ammette soltanto un indirizzamento relativo per determinare la distanza di salto o lo spostamento nella memoria. L'istruzione Bcc (*Branch Conditionally:* salto condizionato) consente di specificare una tra 14 condizioni. Se la condizione selezionata è soddisfatta, l'esecuzione del programma proseguirà dalla locazione di salto designata nell'istruzione. La CPU MC68020 ha anche un'istruzione più complicata, DBcc (*Test Condition, Decrement and Branch:* test di condizione, decremento e salto) che risulta utile per il salto condizionato in una struttura iterativa di programma. Queste istruzioni di salto e quelle che confrontano gli operandi di test sono discusse in questo paragrafo. I valori possibili dei codici di condizione e le istruzioni dell'MC68020 che hanno effetto su di essi sono elencate nell'app. D.

6.2.1 Salti incondizionati: branch e jump

Le istruzioni BRA (*BRanch Always:* salta sempre) e JMP (*JuMP:* salta) producono un trasferimento incondizionato del controllo modificando il valore nel contatore di programma, come mostrato nella Tab. 6.4. Ogniqualvolta avviene un salto, l'indirizzo della nuova istruzione dev'essere su un confine di word — cioè, pari — altrimenti si avrebbe un errore d'indirizzamento. L'istruzione JMP differisce da BRA poiché l'indirizzo di un salto di tipo *jump* può essere specificato da diverse modalità d'indirizzamento, mentre l'indirizzamento in un salto di tipo *branch* è sempre relativo al contatore di programma.

Tab. 6.4 *Istruzioni di salto incondizionato.*

Sintassi	Operazione	Modo d'indirizzo
BRA <spost>	(PC) = (PC) + <spost>	Relativo
JMP <EA>	(PC) = (EA)	Modi di controllo

Note:
1. In BRA <spost>, lo spostamento <spost> è un intero con segno, di 8, 16 o 32 bit.
2. (PC) è la locazione dell'istruzione BRA + 2.
3. I codici di condizione non sono interessati.

L'istruzione BRA. La forma dell'istruzione BRA è:

```
BRA         <spost>
```

che consente uno spostamento di 8 bit, di 16 bit o di 32 bit. Per uno spostamento di 8 bit, l'intervallo di salto si estende da –126 a +129 locazioni di byte, contate a partire dalla locazione dell'istruzione BRA, poiché il valore (PC) è uguale alla locazione dell'istruzione corrente più 2 quando il nuovo valore viene calcolato. Nel caso di uno spostamento di 16 bit, l'intervallo è da –32766 a +32769 byte. Uno spostamento di 32 bit permette di trasferire il controllo di una metà dell'intervallo d'indirizzamento completo dell'MC68020, sia in avanti che all'indietro, rispetto all'istruzione di salto. Il valore dello spostamento è determinato automaticamente dall'assemblatore quando lo spostamento è specificato come un'etichetta della forma seguente:

```
BRA         <etichetta>
```

Le istruzioni BRA e JMP svolgono la medesima funzione, ma la BRA ammette solamente l'indirizzamento relativo al PC con spostamento.

L'istruzione JMP. Questa istruzione ha la forma:

```
JMP         <EA>
```

dove <EA> specifica la locazione dell'istruzione successiva da eseguire. Sono ammesse soltanto le modalità d'indirizzamento di controllo, escludendo così i modi diretto di registro e di autoindicizzazione. Nel modo assoluto, l'istruzione:

```
JMP         $1F000
```

causa il trasferimento del controllo alla locazione $1F000. Si può specificare un indirizzo assoluto di 16 bit o di 32 bit. Se l'indirizzo è specificato dal programma in linguaggio assembler o dall'editor di collegamento (*linkage editor*), la forma dell'istruzione diviene:

```
JMP         ETICH
```

dove ETICH è un simbolo assoluto o rilocabile. La forma assoluta è impiegata normalmente per trasferire il controllo ad una routine definita ad un indirizzo prefissato nel sistema, a causa di considerazioni inerenti all'hardware.

Un indirizzo effettivo in un'istruzione JMP può essere definito anche tramite un indirizzamento indiretto di registro o indiretto di PC. Quando l'indirizzo della routine è definito da un indirizzo base contenuto in un registro d'indirizzo, l'istruzione:

```
JMP         (A1)
```

causa il trasferimento del controllo all'istruzione puntata da (A1). Uno spostamento o un registro indice può essere usato per modificare l'indirizzo base nel registro d'indirizzo, come nell'istruzione di esempio:

```
JMP        (0, A1, D1.L*4)
```

che causa un salto all'indirizzo di longword puntato da (A1) indicizzato da (D1). Il valore dell'indice potrebbe servire a specificare uno di vari punti d'entrata in una routine il cui indirizzo di base è definito in A1. In tutte queste modalità indirette, (An) può essere sostituito da (PC) se è richiesto l'indirizzamento relativo al PC.

Quando gli indirizzi di routine interrelate sono contenuti in una tabella della memoria, sono impiegate le modalità d'indirizzamento indiretto di memoria. Queste modalità consentono di utilizzare direttamente le tabelle di "salto" o di "distribuzione" per dirigere il flusso di controllo entro un programma. Sono ammesse sia il modo indiretto di memoria con registro d'indirizzo che quello indiretto di memoria con PC.

Esempio 6-3

Il segmento di programma in Fig. 6.3 illustra il modo in cui può essere creata ed utilizzata una tabella di salto nella memoria. Un'etichetta JMPTBL definisce il punto di entrata per la porzione eseguibile del programma. Quando viene eseguita, l'istruzione JMP impiega (A0) indicizzato da (D0), che viene scalato (moltiplicato) di 4 per indirizzare un offset di longword da (A0).

Nel programma sono definite due tabelle. Quella definita da TABLE1 contiene l'indirizzo iniziale di quattro routine, ognuna delle quali occupa $100 byte di memoria. La seconda tabella (TABLE2) definisce gli indirizzi assoluti di altre quattro routine o punti d'entrata entro una routine. Per selezionare un indirizzo di salto, l'indirizzo iniziale di una tabella selezionata dev'essere posto in A0 e il numero di entrata (0, 1, 2, ...) in D0, prima dell'esecuzione del segmento di programma. Ciascuna tabella ha la forma seguente:

NUMERO D'ENTRATA	INDIRIZZO	CONTENUTO DELLA MEMORIA
1	(A0)	Primo indirizzo
1	(A0) + 4	Secondo indirizzo
.	.	.
.	.	.
.	.	.
n	(A0) + 4*(D0)	n-esimo indirizzo

```
                          1.          TTL     FIGURA 6.3
                          2.          LLEN    100
00010000                  3.          ORG     $10000
                          4.    *
                          5.    *  ESEMPIO DI TABELLA DI SALTO
                          6.    *     INPUT:  (D0.W) = NUMERO DI ENTRATA
                          7.    *             (A0.L) = INDIRIZZO DI TABELLA
                          8.    *
                          9.    *     PUNTO DI ENTRATA
                          10.   *
                          11.         XDEF    JMPTBL            ;DEFINISCE SIMBOLO GLOBALE
                          12.   *
                          13.   *
00010000  4EF0 0511       14.   JMPTBL JMP    ([0,A0,D0.W*4]) ;SALTA A ENTRATA
                          15.   *
00020000                  16.         ORG     $20000
                          17.   *
    = 00030000            18.   SCHED  EQU    $30000            ;PREPARA I PUNTI DI ENTRATA
    = 00030100            19.   QUEUE  EQU    SCHED+$100        ; DI ESEMPIO
    = 00030200            20.   DISP   EQU    QUEUE+$100
    = 00030300            21.   TIMER  EQU    DISP+$100
                          22.   *
00020000  00030000        23.   TABLE1 DC.L   SCHED             ;TABELLE DI SALTO
00020004  00030100        24.          DC.L   QUEUE
00020008  00030200        25.          DC.L   DISP
0002000C  00030300        26.          DC.L   TIMER
                          27.   *
00020010  00030500        28.   TABLE2 DC.L   $30500            ;INDIRIZZI ASSOLUTI
00020014  00030520        29.          DC.L   $30520
00020018  00030540        30.          DC.L   $30540
0002001C  00030560        31.          DC.L   $30560
                          32.   *
00020020                  33.          END
```

Fig. 6.3 *Esempi di tabelle di salto.*

Il segmento di programma può essere eseguito in vari modi. Per il debugging, si potrebbe utilizzare il monitor per inizializzare (A0) e (D0) e quindi causare il trasferimento del controllo alla locazione JMPTBL. Naturalmente, le istruzioni eseguibili dovrebbero essere poste nelle aree di memoria definite dalle tabelle di salto. Quando il segmento di programma della tabella di salto fa parte di un modulo più grande, le routine che impiegano le tabelle di salto devono essere collegate (tramite linking) col segmento in Fig. 6.3. Il punto di entrata JMPTBL è definito da una direttiva XDEF, che dichiara il simbolo JMPTBL come "pubblico", cioè globale. Un altro modulo che faccia riferimento all'etcihetta JMPTBL deve contenere la direttiva

```
XREF        JMPTBL
```

affinché l'editor di collegamento possa fornire l'indirizzo appropriato per il modulo di caricamento che creerà.

Dopo il salto, il contenuto originale del PC sarà perso, cosicché non sarà possibile fare ritorno nel programma così com'è. Al fine di trasferire il controllo e fare ritorno da un segmento indirizzato dalla tabella, si potrebbe usare una chiamata di subroutine con indirizzamento indiretto per specificare la locazione iniziale. Le subroutine saranno discusse nel par. 6.3.

6.2.2 Salto condizionato

Le istruzioni Bcc consentono di selezionare un percorso di controllo basato sulle condizioni:

IF (condizione verificata)
  THEN (salta alla nuova sequenza)
  ELSE (esegui la prossima istruzione)

La nuova sequenza di istruzioni può essere situata a indirizzi di memoria più alti o più bassi relativamente all'istruzione di salto. Il formato di linguaggio assembler per il salto condizionato generale è:

```
Bcc        <etichetta>
```

che impiega l'indirizzamento relativo al contatore di programma. Lo spostamento aggiunto al valore del contatore di programma per causare il salto può essere un intero con segno di 8 bit, di 16 bit o di 32 bit. Alcuni assemblatori accettano la forma Bcc.S che forza uno spostamento di 8 bit (corto) quando è possibile.

La Fig. 6.4 illustra il funzionamento delle istruzioni Bcc. Le operazioni aritmetiche, come pure un certo numero di altre istruzioni, assegnano i valori ai codici di condizione in base ai risultati della particolare operazione. Se la condizione è verificata, il valore dello spostamento viene aggiunto al valore nel contatore di pro-

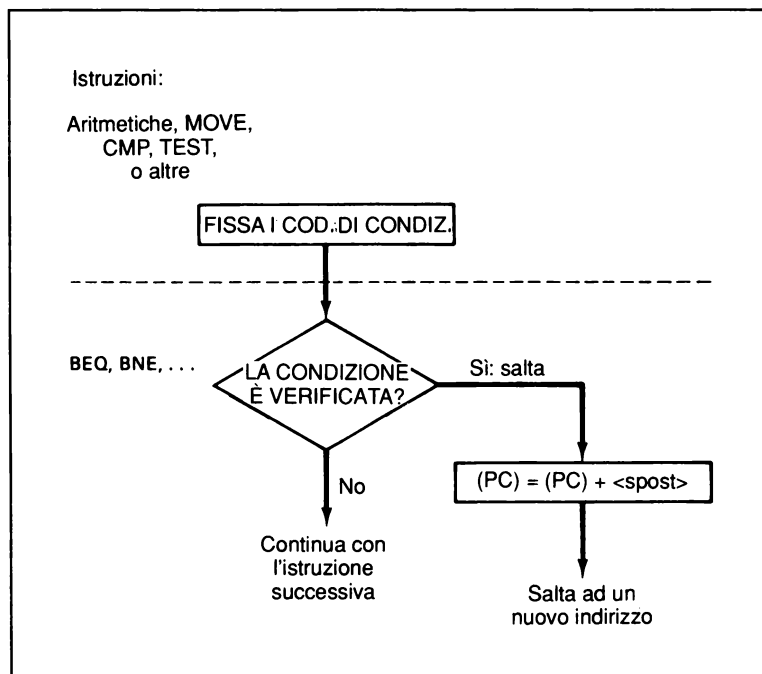



Fig. 6.4
Operazioni delle istruzioni Bcc.

gramma. In quel momento, il PC conterrà l'indirizzo dell'istruzione di salto condizionato più 2. Le condizioni possibili sono elencate nella Fig. 6.5, insieme coi valori dei codici di condizione che causano un salto. Il formato dell'istruzione indica la maniera in cui le istruzioni sono codificate in linguaggio-macchina.

CC	*Carry Clear* (non riporto)	0100	$\overline{C}$
CS	*Carry Set* (riporto)	0101	C
EQ	*Equal* (uguale)	0111	Z
GE	*Greater or Equal* (maggiore o uguale)	1100	$N \cdot V + \overline{N} \cdot \overline{V}$
GT	*Greater Than* (maggiore)	1110	$N \cdot V \cdot \overline{Z} + \overline{N} \cdot \overline{V} \cdot \overline{Z}$
HI	*HIgh* (alto)	0010	$\overline{C} \cdot \overline{Z}$
LE	*Less or Equal* (minore o uguale)	1111	$Z + N \cdot \overline{V} + \overline{N} \cdot V$
LS	*Low or Same* (basso o identico)	0011	$C + Z$
LT	*Less Than* (minore)	1101	$N \cdot \overline{V} + \overline{N} \cdot V$
MI	*MInus* (meno)	1011	N
NE	*Not Equal* (non uguale)	0110	$\overline{Z}$
PL	*PLus* (più)	1010	$\overline{N}$
VC	*oVerflow Clear* (non overflow)	1000	$\overline{V}$
VS	*oVerflow Set* (overflow)	1001	V

Fig. 6.5 *Istruzioni di salto condizionato.*

Codici di condizione: Non interessati.

Formato dell'istruzione:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	Condizione				Spostamento di 8 bit							
Spostamento di 16 bit se spostamento di 8 bit = $00															
Spostamento di 32 bit se spostamento di 8 bit = $FF															

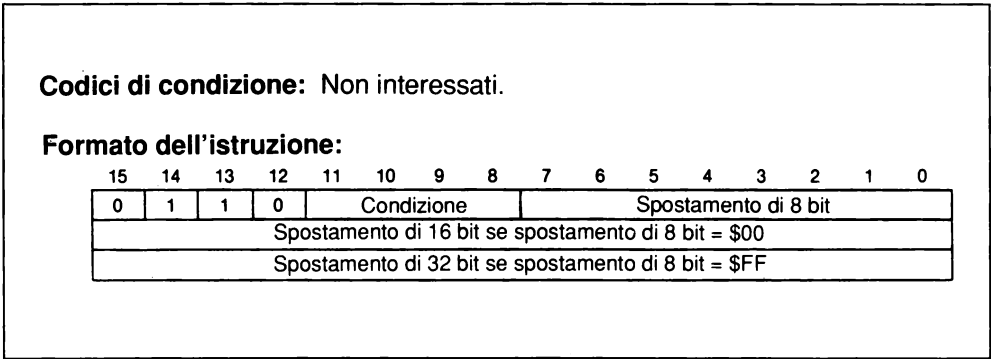



Fig. 6.5 *Istruzioni di salto condizionato. (continuazione)*

Se viene eseguita un'istruzione aritmetica o un'istruzione MOVE, i codici di condizione indicano le condizioni aritmetiche che valgono per l'operando di destinazione. Per esempio, nell'addizione:

ADD.<l>   X,Y

il valore della destinazione, che può essere di lunghezza <l> = B, W o L, è la somma. Il contenuto della destinazione, desginato (Y), può rappresentare un intero con o senza segno. Nel caso di un intero senza segno, il risultato può essere stato zero (Z = {1}) o diverso da zero (Z = {0}). Se la somma è troppo grande, essa è indicata da un riporto (C = {1}). Un'operazione aritmetica con segno potrebbe produrre una condizione di fuori-intervallo (V = {1}) o una somma nulla, positiva o negativa. I possibili test per gli interi con e senza segno sono elencati nella Tab. 6.5.

Tab. 6.5 *Test condizionali.*

Istruzioni	**Condizioni per il salto**	
	Senza segno	*Con segno*
ADD.<l> X,Y o SUB.<l> X,Y o MOVE.<l> X,Y	(Y) = 0 BEQ (Y) $\neq$ 0 BNE	(Y) = 0 BEQ (Y) $\neq$ 0 BNE (Y) $\geq$ 0 BPL (Y) < 0 BMI
Fuori-intervallo per istruzioni aritmetiche	C = {1} BCS C = {0} BCS	V = {1} BVS V = {0} BVC

Nota: Per l'istruzione MOVE operante su interi con segno, sono ammesse anche le condizioni di salto BGE, BLT, BGT e BLE.

Dopo un'istruzione MOVE, possono essere esaminati i codici di condizione, ma non è possibile alcuna condizione di fuori-intervallo. I codici di condizione C e V sono azzerati, per cui C = {0} e V = {0} dopo il trasferimento. Come indicato nella tabella, tranne che per i test di zero o diverso da zero, i test di condizione applicabili sono diversi a seconda che l'intero sia con segno o senza segno. Questo paragrafo introduce le condizioni di salto richieste in entrambi i casi. Ulteriori discussioni delle tecniche di programmazione per le operazioni aritmetiche saranno presentate nel cap. 7.

Salto se zero o se diverso da zero. Le istruzioni di salto condizionato BEQ e BNE sono logicamente opposte. Dopo che un'istruzione precedente ha fissato i codici di condizione, l'istruzione

```
BEQ        <etichetta>
```

produrrà un salto se il risultato era zero. Un salto su una condizione di risultato diverso da zero avrebbe richiesto l'istruzione:

```
BNE        <etichetta>
```

Entrambe le istruzioni sono ammesse per operazioni aritmetiche su interi sia con segno che senza segno, e rappresentano gli unici salti condizionati che possono essere usati indipendentemente dalla presenza del segno.

Salti con operazioni aritmetiche su interi senza segno. L'aritmetica senza segno opera sugli interi positivi e sullo zero. Gli indirizzi dovrebbero essere trattati come numeri senza segno. Gli unici codici di condizione che dovrebbero essere esaminati dopo un'addizione o una sottrazione di interi senza segno sono lo zero (Z) e il riporto (*carry :* C).

Il codice di condizione C = {1} indica che un riporto è stato generato da un'addizione, poiché la somma era troppo grande per la lunghezza specificata dell'operando. Nella sottrazione, un bit di riporto posto a {1} rappresenta un riporto negativo (*borrow*) poiché il sottraendo era maggiore del minuendo. In entrambi i casi, il risultato non è un valido intero senza segno. Quando vengono eseguite operazioni aritmetiche su interi senza segno, si può usare l'istruzione di test per il salto su riporto avvenuto (*Branch on Carry Set:* BCS) o su riporto non avvenuto (*Branch on Carry Clear:* BCC), per selezionare i percorsi nel programma.

L'istruzione Bcc è impiegata in un programma perlopiù per creare dei cicli che eseguano le parti iterative dell'algoritmo in questione. Per esempio, la forma più semplice di iterazione si ha quando il numero di ripetizioni è noto. In vari esempi di programmazione svolti in precedenza, la terminazione di un ciclo ("LOOP") si basava sul valore di un contatore, che veniva decrementato progressivamente fino a zero, a partire dal numero di iterazioni richieste. Quindi, l'istruzione BNE LOOP che

seguiva l'istruzione di decremento (SUB #1,Dn nella maggior parte dei casi) causava la terminazione del ciclo allorché (Dn) = 0. Se l'istruzione di salto è l'ultima espressione nel ciclo, questo viene definito con *post-test* ed assume la forma:

```
REPEAT
        (corpo del ciclo)
UNTIL (conto = 0)
```

In un ciclo siffatto, le istruzioni che compongono il corpo del ciclo vengono eseguite almeno una volta. La sequenza di istruzioni FORTRAN:

```
        DO 10  I = 1,20
        (corpo del ciclo)
10      CONTINUE
```

costituisce un ciclo di questo tipo. Altri tipi di strutture cicliche sono mostrati in numerosi esempi del libro.

Esempio 6-4

Il programma mostrato nella Fig. 6.6 somma due array di vettori unidimensionali di interi con segno di 16 bit. Il programma calcola X(I) + Y(I) e salva la somma in X(I) per ciascun elemento negli array. La lunghezza degli array è inserita in D0 e gli indirizzi degli array devono essere posti in A0 e A1 prima che il programma sia eseguito. Dopo ogni addizione, viene esaminato il codice di condizione C. Se il bit C vale {1}, allora è avvenuto un overflow e la word meno significativa di D0 conterrà il valore $FFFF.

```
                        1.          TTL     FIGURA 6.6
                        2.          LLEN    100
00010000                3.          ORG     $10000
                        4.  *
  = 00000063            5.  RETURN  EQU     $0063                   ;INDIRIZZO DEL MONITOR
                        6.  *
                        7.  *   CALCOLA SOMME DI ARRAY INTERI SENZA SEGNO
                        8.  *       INPUT:  (A0.L) = ARRAY X DI NUMERI DI 16 BIT
                        9.  *               (A1.L) = ARRAY Y DI NUMERI DI 16 BIT
                       10.  *               (D0.W) = NUMERO DI ELEMENTI IN ARRAY
                       11.  *       OUTPUT: (D0.W) = $FFFF IN CASO DI ERRORE
                       12.  *
00010000  48E7 60C0    13.  SUM     MOVEM.L D1-D2/A0-A1,-(SP)       ;SALVA I REGISTRI
00010004  3200         14.          MOVE.W  D0,D1                   ;PREDISPONE IL CONTATORE
00010006  6700 0012    15.          BEQ     ERROR                   ;SE ZERO, ESCE
                       16.  *                                       ;  CON ERRORE
                       17.  *
0001000A  5341         18.  LOOP    SUBQ.W  #1,D1                   ;DECREMENTA INDICE
0001000C  6B00 0010    19.          BMI     FINE                    ;ESCE QUANDO (D1) < 0
00010010  3431 1200    20.          MOVE.W  (0,A1,D1.W*2),D2        ;PRENDE IL VALORE DI Y
00010014  D570 1200    21.          ADD.W   D2,(0,A0,D1.W*2)        ;SOMMA X := X + Y
                       22.  *
00010018  64 F0        23.          BCC     LOOP                    ;SE NON OVERFLOW, RICICLA
                       24.  *                                       ; ALTRIMENTI
0001001A  303C FFFF    25.  ERROR   MOVE.W  #$FFFF,D0               ;LO STATO INDICA L'ERRORE
                       26.  *
0001001E  4CDF 0306    27.  FINE    MOVEM.L (SP)+,D1-D2/A0-A1       ;RIPRISTINA I REGISTRI
00010022  4E4F         28.          TRAP    #15                     ; E RITORNA
00010024  0063         29.          DC.W    RETURN
00010026               30.          END
```

Fig. 6.6 *Verifica di overflow per aritmetica senza segno.*

Nel programma, (D1) è impiegato come contatore ed anche come un valore di indice negli array. Dopo che D1 è stato caricato col conto di word e confrontato con zero, il valore in D1 viene scalato di 2 per fornire il conto di word ad ogni iterazione del ciclo. All'intero del ciclo, il contatore e l'indice in D1 sono decrementati di 1 ed esaminati prima che venga eseguita ciascuna addizione. Viene usata l'istruzione BMI (*Branch if MInus:* salta se negativo) in modo che l'ultima istruzione sia eseguita quando (D1) = 0. S'impiega l'indirizzamento indiretto di registro d'indirizzo con indice per selezionare i valori da sommare. Si noti che gli ultimi elementi negli array sono sommati per primi, poiché (D1) inizia nel ciclo col (conto meno 1) come un indice e viene decrementato fino a zero. Se non avviene alcun overflow, l'istruzione BCC riporta il controllo alla prima istruzione nel ciclo; altrimenti, viene indicata una condizione di errore.

Salti con operazioni aritmetiche con segno. Se i valori da trattare sono numeri in complemento a 2, allora sono applicabili i codici di condizione N, Z e V. Dopo le operazioni aritmetiche, V = {1} indica una condizione di fuori-intervallo. L'istruzione BVS produce un salto quando V = {1}, mentre BVC causa un salto quando V = {0}. Se il risultato è valido, allora l'operando può essere esaminato per vedere se contiene un valore zero, o diverso da zero, o positivo, o negativo. Dopo un'istruzione MOVE, V = {0}, e anche BGE, BLT, BGT e BLE eseguono dei test validi secondo le equazioni logiche dalla Fig. 6.5. Quando V = {0}, BGE ha il medesimo effetto di BPL.

6.2.3 Salto dopo CMP o TST

Le istruzioni CMP (*CoMPare:* confronto) e TST (*TeST*) sono usate per assegnare i valori ai codici di condizione in base ai valori degli operandi. Dopodiché, si possono usare le istruzioni di salto condizionato per dirigere il flusso di controllo nel programma. L'istruzione

```
CMP      X,Y
```

confronta due operandi eseguendo il calcolo

(Y) – (X)

per definire i codici di condizione N, Z, V e C. L'istruzione

```
TST      Y
```

valuta un operando eseguendo il calcolo (Y) – 0, che azzera sempre V e C ma fissa N e Z in base al risultato. Entrambe queste istruzioni definiscono i codici di condizione ma non modificano gli operandi. Le istruzioni discusse in questo paragrafo

sono elencate nella Tab. 6.6. L'istruzione CMP ha le varianti di confronto immediato (*CoMPare Immediate:* CMPI) e di confronto di memoria (*CoMPare Memory:* CMPM), come mostrato.

L'istruzione di confronto:

```
CMP.<l>     <EA>,<Dn>
```

sottrae l'operando di sorgente dal contenuto del registro di dati specificato. Il calcolo

(Dn) – (EA)

viene eseguito senza modificare l'operando; la lunghezza <l> di ciascun operando può essere definita come B, W o L. Se la sorgente è un registro d'indirizzo, la lunghezza dell'operando è ristretta a word (W) o longword (L).

L'istruzione di confronto immediato CMPI confronta un valore immediato con un operando riferito da una modalità d'indirizzamento di dati. Ciò esclude la modalità d'indirizzamento diretto di registro d'indirizzo. Sono ammessi operandi di byte, word o longword. Per esempio, l'istruzione:

```
CMPI.B      #5,$2000
```

confronta il valore 5 col contenuto del byte alla locazione $2000. Diversamente dall'istruzione CMP, l'istruzione CMPI può far riferimento a locazioni di memoria come destinazione.

L'istruzione di confronto di memoria CMPM è usata per confrontare sequenze di byte, di word o di longword nella memoria. Soltanto il modo d'indirizzamento con postincremento è ammesso per entrambi gli operandi. L'istruzione

```
CMPM.B      (A1)+,(A2)+
```

confronta i byte indirizzati da A1 e A2 e poi incrementa gli indirizzi per puntare ai byte successivi.

L'istruzione di test TST ha il formato:

```
TST <l>     <EA>
```

dove <l> può essere B, W o L, mentre <EA> può essere specificato da qualsiasi modo d'indirizzamento tranne quello diretto di registro d'indirizzo. L'istruzione imposta i codici di condizione Z e N in base al valore dell'operando. Comunque, V e C sono sempre azzerati.

Tab. 6.6 *Istruzioni di confronto e di test.*

Istruzione	Sintassi		Modi d'indirizzamento Sorgente	Destinazione	Operazione	Codici di condizione interessati
Confronto	CMP.<l>	<EA>,<Dn>	Tutti	<Dn>	(Dn) – (EA)	N, Z, V, C
Confronto immediato	CMPI.<l>	#<d>,<EA>	d	Dato	(EA) – d	N, Z, V, C
Confronto di memoria	CMPM.<l>	(Am)+,(An)+	(Am)+	(An)+	((An)) – ((Am))	N, Z, V, C
Test	TST.<l>	<EA>	——	Dato	(EA) – 0	N, Z, C = {0}, V = {0}

Note:

1. <l> denota B; W o L
2. Se An è la sorgente per CMP, sono ammessi soltanto operandi di word (W) o di longword (L).

I salti condizionati ammessi dopo l'istruzione TST per un operando intero senza segno sono BEQ e BNE. Queste condizioni per un valore zero o diverso da zero sono anche ammesse per gli interi con segno. Poiché l'istruzione TST azzera i codici di condizione C e V, è ammesso anche un certo numero di altri test per interi con segno. Dopo l'istruzione

```
TST      X
```

le istruzioni di salto condizionato BGT, BLT o BMI, BGE o BPL e BLE possono essere usate quando la locazione X contiene un intero con segno.

Quando un'istruzione Bcc segue un'istruzione CMP nella sequenza:

```
CMP      X,Y
Bcc      <etichetta>
```

il confronto effettuato è il seguente:

```
IF (Y) "condizione cc" (X)
        THEN salta a <etichetta>
        ELSE continua
```

Per esempio, la sequenza di istruzioni:

```
CMP.W    D1,D2
BGE      FINE
```

verifica se il valore di 16 bit in D2 è maggiore o uguale al valore di 16 bit in D1. In caso affermativo, viene eseguito il salto a FINE.

L'istruzione TST confronta un operando con zero in maniera simile. Quindi la sequenza:

```
TST.W    D2
BEQ      FINE
```

ha la logica seguente:

```
IF (D2)[15:0] uguale 0
        THEN salta a FINE
        ELSE continua
```

La Tab. 6.7 elenca le istruzioni Bcc che causerebbero un salto se eseguite dopo l'istruzione CMP o TST. I test per BLT (*Branch on Less Than:* salta se minore) e BMI (*Branch on MInus:* salta se negativo), come pure su BGE e BPL, sono equivalenti dopo un'istruzione TST poiché il bit di overflow V è azzerato.

Tab. 6.7 *Istruzioni Bcc con CMP e TST.*

Istruzione	Risultato	Condizione di salto	
		Senza segno	**Con segno**
CMP X,Y	(Y) = (X) (Y) ≠ (X) (Y) > (X) (Y) ≥ (X) (Y) < (X) (Y) ≤ (X)	BEQ (uguale) BNE (non uguale) BHI (alto) BCC nessun (riporto) BCS (riporto) BLS (basso o stesso)	BEQ (uguale) BNE (non uguale) BGT (maggiore) BGE (maggiore o uguale) BLT (minore) BLE (minore o uguale)
TST X	(X) = 0 (X) ≠ 0 (X) > 0 (X) < 0 (X) ≥ 0 (X) ≤ 0	BEQ BNE BNE --- --- ---	BEQ BNE BGT BLT, BMI BGE, BPL BLE

Note:
1. In CMP X,Y, la destinazione è un registro di dati.
2. TST definisce C = {0} e V = {0}.
3. BMI (*Branch on MInus:* salta se meno) coincide con BLT, e BPL (*Branch on PLus:* salta se più) coincide con BGE, quando V = {0}.

Sia CMP che TST possono essere usate con interpretazioni su interi con segno e senza segno. Tuttavia, il processore esegue sempre il calcolo nell'aritmetica in complemento a 2. Pertanto, le istruzioni di salto condizionato ammissibili sono differenti per le due interpretazioni degli interi.

Esempio 6-5

Il programma di esempio nella Fig. 6.7 confronta due tabelle di byte indirizzate da A1 e A2. Il registro D1 contiene il numero di byte in ciascuna tabella, mentre D2 contiene una word di stato o "flag" per indicare se le tabelle sono identiche. Il flag è inizializzato ad un valore prefissato nullo, per indicare valori diversi prima dell'inizio del confronto. Se tutti i byte corrispondenti delle due tabelle sono uguali, allora il valore +1 viene inserito in D2 dall'ultima istruzione prima di tornare al programma di monitor. Se il numero di byte è zero, D2 sarà lasciato con un valore inizializzato a zero. Tale programma è impiegato di solito per confrontare due stringhe di caratteri ASCII per determinare se sono uguali.

```
                          1.         TTL     FIGURA 6.7
                          2.         LLEN    100
00010000                  3.         ORG     $10000
                          4.  *
  = 00000063              5.  RETURN EQU     $0063            ;MONITOR
                          6.  *
                          7.  *  CONFRONTA DUE TABELLE DI BYTE
                          8.  *  INPUT:        (A1.L) = INDIRIZZO DELLA PRIMA TABELLA
                          9.  *                (A2.L) = INDIRIZZO DELLA SECONDA TABELLA
                          10. *                (D1.B) = NUMERO DI BYTE NELLE TABELLE
                          11. *                         (255 AL MASSIMO)
                          12. *
                          13. *  OUTPUT:       (D2.W) = 0 : NON UGUALE ;STATO
                          14. *                       = 1 : UGUALE
                          15. *
00010000  48E7 4060       16. COMPARE MOVEM.L D1/A1-A2,-(SP)  ;SALVA I REGISTRI
00010004  4242            17.         CLR.W   D2              ;DEFINISCE LO STATO NORMALE
00010006  4A01            18.         TST.B   D1              ;SE LA LUNGHEZZA E' ZERO,
00010008  6700 0010       19.         BEQ     FINE            ; ALLORA SALTA A FINE
                          20. *
0001000C  B509            21. LOOP    CMPM.B  (A1)+,(A2)+     ;ALTRIMENTI, SE DUE BYTE
                          22. *                               ; NON SONO UGUALI,
0001000E  6600 000A       23.         BNE     FINE            ;ALLORA SALTA A FINE
00010012  5301            24.         SUBQ.B  #1,D1           ;ALTRIMENTI DECREMENTA IL CONTATORE
00010014  66 F6           25.         BNE     LOOP            ; ED ESAMINA IL VALORE SUCCESSIVO
                          26. *                               ; FINCHE' (D1) = 0.
                          27. *
00010016  343C 0001       28. EQUAL   MOVE.W  #1,D2           ;ESSI SONO IDENTICI
                          29. *                               ;  PONE STATO = 1
                          30. *
0001001A  4CDF 0602       31. FINE    MOVEM.L (SP)+,D1/A1-A2  ;RIPRISTINA I REGISTRI
0001001E  4E4F            32.         TRAP    #15             ; E RITORNA
00010020  0063            33.         DC.W    RETURN
                          34. *
00010022                  35.         END
```

Fig. 6.7 *Uso delle istruzioni CMP e TST.*

Prima del confronto, un'istruzione TST determina se D1 ha un conto di byte diverso da zero. Entro il ciclo, i byte indirizzati da A1 e da A2 sono confrontati e gli indirizzi vengono incrementati automaticamente. Se due byte qualsiasi sono diversi, la prima istruzione BNE causa un salto e il test termina. Dopo ogni confronto riuscito, il valore del contatore in D1 viene decrementato. Il ciclo viene ripetuto finché il contatore non raggiunge lo zero. Se tutti i byte sono uguali, D2 sarà posto a 1 dall'istruzione all'etichetta LABEL. Prima che il programma possa essere eseguito, A1, A2 e D1 devono essere inizializzati con i valori d'ingresso appropriati.

Esempio 6-6

Il programma mostrato nella Fig. 6.8 confronta interi positivi di 16 bit in un array unidimensionale o tabella memorizzata nelle locazioni NUM1, NUM1 + 2, e così via, e lascia il valore massimo in D1. Il registro D3 è usato come contatore per determinare la condizione in cui tutti i numeri sono stati esaminati. Si suppone che D3 contenga il conto dei numeri prima dell'esecuzione del programma.

```
                         1.         TTL     FIGURA 6.8
                         2.         LLEN    100
00010000                 3.         ORG     $10000
                         4.   *
  # 00000063             5.   RETURN EQU    $0063           ;MONITOR
                         6.   *
                         7.   *  TROVA IL MASSIMO INTERO DI 16 BIT NELLA TABELLA NUM1
                         8.   *
                         9.   *  INPUT:      (D3.W) = NUMERO DI INTERI DA ESAMINARE
                        10.   *                NUM1 = TABELLA DI INTERI (AL MASSIMO, 40)
                        11.   *  OUTPUT:     (D1.W) = MASSIMO INTERO TROVATO
                        12.   *
00010000 48E7 3040      13.         MOVEM.L D2-D3/A1,-(SP)  ;SALVA  I REGISTRI
00010004 227C 00010024  14.         MOVE.L  #NUM1,A1        ;CARICA INDIRIZZO INIZIALE
0001000A 5343           15.         SUBQ.W  #1,D3           ;NUMERO DI INTERI - 1
0001000C 3219           16.         MOVE.W  (A1)+,D1        ;(D1) = PRIMO INTERO
                        17.   *
0001000E 3419           18.   LOOP  MOVE.W  (A1)+,D2        ;PRENDE L'INTERO SUCCESSIVO
00010010 B441           19.         CMP.W   D1,D2           ;CONFRONTA COL MASSIMO
00010012 6300 0004      20.         BLS     NEXT            ; SE MINORE O UGUALE
                        21.   *                             ;   ESAMINA IL SUCCESSIVO
00010016 C342           22.         EXG     D1,D2           ;ALTRIMENTI: SCAMBIA
                        23.   *
00010018 5343           24.   NEXT  SUBQ.W  #1,D3           ;DECREMENTA IL CONTATORE
0001001A 66 F2          25.         BNE     LOOP            ; RICICLA FINCHE' (D3) = 0
                        26.   *
0001001C 4CDF 020C      27.         MOVEM.L (SP)+,D2-D3/A1  ;RIPRISTINA I REGISTRI
00010020 4E4F           28.         TRAP    #15             ; E RITORNA
00010022 0063           29.         DC.W    RETURN
                        30.   *
00010024 <50>           31.   NUM1  DS.L    20              ;NUMERI
                        32.   *
00010074                33.         END
```

Fig. 6.8 *Confronto di interi senza segno.*

Poiché sono considerati soltanto numeri positivi, viene usata l'istruzione BLS (*Branch if Lower or Same:* salta se più basso o indentico) per determinare se il valore in D2 è minore del massimo presunto, contenuto in D1. In caso affermativo, viene eseguito un test sul completamento del ciclo. Se il nuovo valore in D2 è maggiore del "massimo" in D1, i contenuti dei registri vengono scambiati. Nell'esempio, la direttiva DS (*Define Storage:* riserva memoria) riserva lo spazio di memoria per 40 interi di 16 bit. I valori devono essere memorizzati in queste locazioni prima che il programma sia eseguito.

6.2.4 L'istruzione DBcc

L'istruzione DBcc di test, decremento e salto è particolarmente efficiente per il controllo di strutture cicliche. Il formato fondamentale del linguaggio assembler è

```
DBcc        <Dn>,<etichetta>
```

che designa tre parametri: la condizione "cc", un registro di dati e uno spostamento. Quest'ultimo è rappresentato qui come un'etichetta in un programma in linguaggio assembler. Le varie condizioni ed i formati per le istruzioni sono mostrate nella Fig. 6.9.

CC	*Carry Clear* (non riporto)	0100	$\overline{C}$
CS	*Carry Set* (riporto)	0101	C
EQ	*Equal* (uguale)	0111	Z
F	*never true* (mai vero)	0001	0
GE	*Greater or Equal* (maggiore o uguale)	1100	$N \cdot V + \overline{N} \cdot \overline{V}$
GT	*Greater Than* (maggiore)	1110	$N \cdot V \cdot \overline{Z} + \overline{N} \cdot \overline{V} \cdot \overline{Z}$
HI	*HIgh* (alto)	0010	$\overline{C} \cdot \overline{Z}$
LE	*Less or Equal* (minore o uguale)	1111	$Z + N \cdot \overline{V} + \overline{N} \cdot V$
LS	*Low or Same* (basso o identico)	0011	$C + Z$
LT	*Less Than* (minore)	1101	$N \cdot \overline{V} + \overline{N} \cdot V$
MI	*MInus* (meno)	1011	N
NE	*Not Equal* (non uguale)	0110	$\overline{Z}$
PL	*PLus* (più)	1010	$\overline{N}$
T	always True (sempre vero)	0000	1
VC	*oVerflow Clear* (non overflow)	1000	$\overline{V}$
VS	*oVerflow Set* (overflow)	1001	V

Fig. 6.9 *L'istruzione DBcc.*

Codici di condizione: Non interessati.

Formato dell'istruzione:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	Condizione				1	1	0	0	1	Registro		
Spostamento															

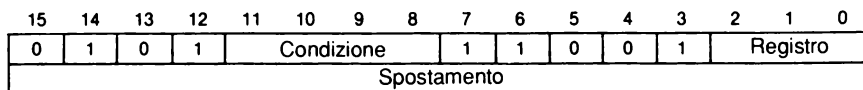



Fig. 6.9 *L'istruzione Dbcc. (continuazione)*

L'istruzione DBcc può causare la terminazione di un ciclo quando si verifica la condizione specificata (cc) o quando il conto in <Dn> raggiunge il valore −1 in seguito al decremento. Il grafo di flusso per l'istruzione in Fig. 6.10 illustra le condizioni che causano l'esecuzione dell'istruzione successiva nella sequenza.

Quattordici delle condizioni logiche esaminate da DBcc coincidono con quelle per l'istruzione Bcc discusse in precedenza. Tuttavia, DBcc è talvolta denotata come istruzione di "non salto su condizione": cioè, se la condizione è vera, non viene effettuato alcun salto, che è l'azione opposta a quella svolta dall'istruzione Bcc. Una struttura ciclica con DBcc che termina il ciclo ha la logica seguente:

```
REPEAT
        (corpo del ciclo)
UNTIL (condizione)
```

Per esempio, la sequenza di istruzioni con la struttura:

```
LOOP ...
            (corpo del ciclo)
TST         X               ; TEST PER ZERO
DBEQ        <Dn>,LOOP       ; RIPETE IL CICLO SE NON ZERO
```

continuerà a ripetere il ciclo finché il contenuto della locazione X non sarà divenuto zero o finché non sarà stato esaurito il conto in Dn. L'istruzione TST definisce i codici di condizione in base al valore (X). L'istruzione DBcc esamina le condizioni e decrementa <Dn> se la condizione è falsa ma non altera i codici di condizione. Il ciclo termina quando la condizione specificata è vera o quando <Dn> contiene −1. Se il programma di esempio in questione necessitasse di eseguire un ciclo (salto) in base alla condizione X = 0, si dovrebbe usare la condizione logica opposta, DBNE. Allora, se X è 0, viene effettuato un salto a LOOP; altrimenti, l'esecuzione continua dall'istruzione successiva.

DBcc <Dn>,<ETICH>

CONDIZIONE cc VERIFICATA?

Sì: eseguito

No: decrementa il contatore ed esegui il test

(Dn) ← (Dn) −1

(Dn) = −1 ?

Sì: conto completo

No

(PC) ← (PC) + <spost>

(PC) ← (PC) +2

USCITA da DBcc

Salta a <ETICH>

Esegui l'istruzione successiva in sequenza

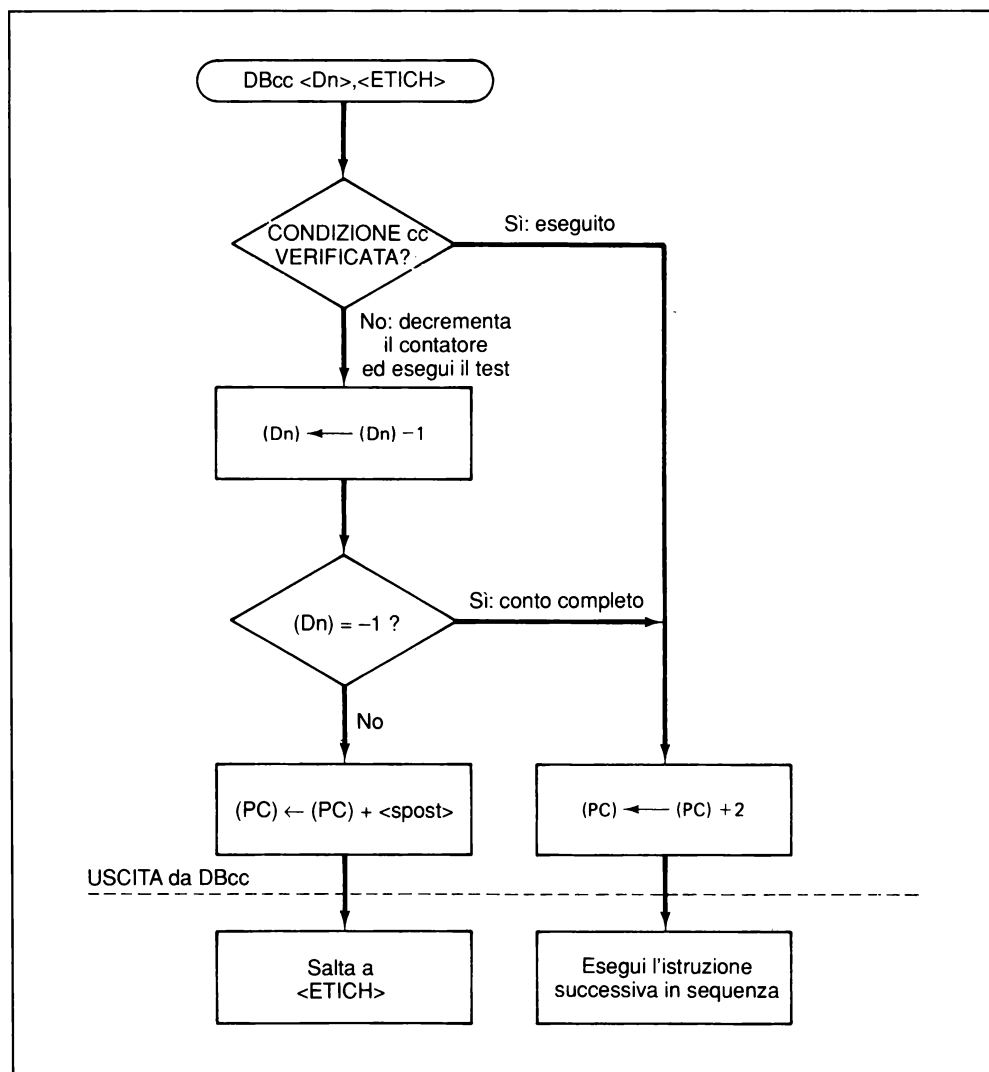



Fig. 6.10 *Le operazioni dell'istruzione DBcc.*

Oltre alle 14 condizioni di test che possono essere usate da entrambe le istruzioni DBcc e Bcc, la DBcc ha due altre condizioni, T (*True:* vero) e F (*False:* falso). L'istruzione DBT non produce mai un salto; il suo opposto logico, DBF, causa sempre un salto a meno che il conteggio non sia esaurito. Per l'istruzione DBF, non viene esaminata alcuna condizione. L'istruzione DBF sostituisce la sequenza di "decremento e test per zero" usata spesso per terminare un ciclo.[1] Tuttavia, c'è una differenza: poiché il contatore deve raggiungere −1 prima che la ripetizione del ciclo termini, il valore iniziale del contatore dev'essere inferiore di uno rispetto al numero di iterazioni richieste.

[1] Alcuni assemblatori dell'MC68020 usano la forma "DBRA" anziché il codice mnemonico DBF.

Il registro specificato per tenere il conto contiene un intero di 16 bit con un valore decimale compreso tra 0 e 65535. Supponendo che il registro contenesse inizialmente il valore intero N e che anche la condizione cc non è vera, allora in un ciclo con post-test il conto sarebbe N, N – 1, N – 2, ..., 0 prima di uscire dal ciclo; quindi saranno eseguite N + 1 iterazioni. Per eseguire il ciclo N volte, il registro del contatore deve contenere inizialmente il valore N – 1.

Esempio 6-7

I due brevi segmenti di programma nella Fig. 6.11 confrontano l'impiego delle istruzioni Bcc e DBcc. Ciascun programma esamina una tabella contenente (N + 1) operandi lunghi una word per individuare un elemento diverso da zero. Prima dell'esecuzione, il registro d'indirizzo A1 deve contenere il primo indirizzo della tabella mentre D1 contiene N. Nel primo segmento, un elemento diverso da zero indirizzato da (A1) causa un salto all'etichetta DONE1. L'indirizzo dell'elemento diverso da zero è (A1) – 2, poiché l'indirizzamento con post-incremento è impiegato nell'istruzione TST. Se non viene trovato alcun elemento diverso da zero, (D1) viene decrementato finché non avrà raggiunto –1; a quel punto l'istruzione BPL porrà fine al ciclo. Quindi, se (D1) = –1 dopo che il segmento di programma è stato completato, la tabella conterrà valori tutti nulli.

```
                         1.          TTL     FIGURA 6.11
                         2.          LLEN    100
00010000                 3.          ORG     $10000
                         4.  *
    # 00000063           5.  RETURN  EQU     $0063            ;MONITOR
                         6.  *
                         7.  *   CONFRONTO DELLE OPERAZIONI DBCC E BCC
                         8.  *     INPUT:    (D1.W) = LUNGHEZZA DI TABELLA - 1
                         9.  *               (A1.L) = INDIRIZZO DI TABELLA
                        10.  *
                        11.  *     OUTPUT:   (A1.L) = INDIRIZZO DI ENTRATA NON NULLA
                        12.  *               (D1.W) = $FFFF : NESSUNA ENTRATA NON NULLA
                        13.  *
                        14.  *  OPERAZIONE DI BCC
                        15.  *
00010000  4A59          16.  LOOP1   TST.W   (A1)+
00010002  6600 0006     17.          BNE     FINE1            ;TERMINA SE NON ZERO
                        18.  *                                ;  ALTRIMENTI, TEST VALORE SUCC.
00010006  5341          19.          SUBQ.W  #1,D1
00010008  6A F6         20.          BPL     LOOP1            ;TEST FINCHE' CONTATORE E' -1
0001000A  5589          21.  FINE1   SUBQ.L  #2,A1            ;INDIRIZZO DI UN VALORE NON ZERO
                        22.  *                                ;  SE TROVATO
0001000C  4E4F          23.          TRAP    #15
0001000E  0063          24.          DC.W    RETURN
                        25.  *
00020000                26.          ORG     $20000
                        27.  *
                        28.  *  OPERAZIONE DI DBCC
                        29.  *
00020000  4A59          30.  LOOP2   TST.W   (A1)+
00020002  56C9 FFFC     31.          DBNE    D1,LOOP2         ;RICICLA SE IL VALORE E' ZERO
                        32.  *                                ;  O SE CONTATORE > -1
                        33.  *                                ; ALTRIMENTI: VA A ISTRUZ. SUCC.
                        34.  *
00020006  5589          35.  FINE2   SUBQ.L  #2,A1            ;INDIRIZZO DEL PRIMO VALORE
                        36.  *                                ;  DIVERSO DA ZERO
                        37.  *
00020008  4E4F          38.          TRAP    #15              ;RITORNA AL MONITOR
0002000A  0063          39.          DC.W    RETURN
0002000C                40.          END
```

Fig. 6.11 *Confronto di interi senza segno.*

L'istruzione per determinare se un valore è diverso da zero e quelle che effettuano il decremento e il test sul conteggio del ciclo nel primo programma possono essere sostituite da una singola istruzione DBcc. Il secondo segmento nella Fig. 6.11 mostra le istruzioni per eseguire il test di "non zero" sulle (N - 1) locazioni, come prima.

ESERCIZI

6.2.1 Si discuta l'impiego di una tabella di salto se il programma è in una memoria a sola lettura, ma le routine da eseguire possono avere differenti indirizzi iniziali in sistemi diversi.

6.2.2 Si confronti l'uso dell'indirizzamento indiretto nelle seguenti istruzioni:

```
(a) JMP         (A2)
(b) MOVEA       (A2),A1
(c) MOVE.W      (0,D1.W), D2
(d) JMP         ([A2])
```

6.2.3 Se è stata eseguita l'istruzione

```
CMP.W    I,J
```

si specifichi l'istruzione che produrrà l'azione seguente:

(a) salto a ZERO se I = J
(b) salto a MINORE se I < J
(c) salto a MAGGIORE se I > J

dove ZERO, MINORE e MAGGIORE sono etichette di istruzioni.

6.2.4 Si riscriva la sequenza:

```
LOOP ...
          (corpo del ciclo)
          SUB.W #1,CNT
          BNE   LOOP
```

impiegando un'istruzione DBcc.

6.2.5 Si scriva il programma in linguaggio assembler che svolge la medesima funzione del seguente ciclo di FORTRAN:

```
          DO 10 I = 1,20
                (corpo del ciclo)
10        CONTINUE
```

6.2.6

Si scriva il programma in linguaggio assembler equivalente alla sequenza di istruzioni riportata di seguito:

```
        SUM = 0
        I = MAXVAL
   10   SUM = SUM + 1
        I = I - 1
        IF (I .NE. 0) GOTO 10
```

Si supponga che il valore di MAXVAL sia stato assegnato in precedenza. Si esegua un test del programma per i tre valori MAXVAL = 10, 1 e 0. Si modifichi il programma per tener conto del caso in cui MAXVAL = 0.

6.2.7

Poiché le istruzioni MOVE e TST pongono sempre il codice di condizione V = {0}, si dimostri che BGE, BLT, BGT e BLE producono salti validi dopo un'istruzione MOVE o TST quando vengono trasferiti interi con segno. Si dimostri anche che BGE coincide con BPL e che BLT coincide con BMI in questo caso.

6.2.8

Si determinino i codici di linguaggio-macchina per ciascuna delle seguenti istruzioni di salto:

Indirizzo	*Etichetta*	*Istruzione*
10000	CICLO1	BRA.B CICLO2
.	.	.
.	.	.
1006E	CICLO2	BRA.W CICLO3
.	.	.
.	.	.
14000	CICLO3	BRA.L CICLO4
.	.	.
.	.	.
10000	CICLO4	BRA.L CICLO1

6.3 IMPIEGO DELLE SUBROUTINE CON L'MC68020

Una *subroutine* è una sequenza di istruzioni che sono trattate come un modulo di programma a sé entro un programma più grande. Una subroutine può essere "chiamata" o eseguita una o più volte durante l'esecuzione del programma principale. Generalmente, le subroutine associate con un programma eseguono compiti specifici, ciascuno dei quali rappresenta una procedura più semplice di quella dell'intero programma. In effetti le subroutine sono denominate *procedure* nel linguaggio Pascal. Ogni modulo (o singola subroutine) dovrebbe essere "autonomo", cioè esaminabile indipendentemente dal programma principale. Quando la subroutine viene chiamata durante l'esecuzione di un programma, le sue istruzioni vengono eseguite e poi il controllo viene riportato all'istruzione in sequenza che segue la chiamata della subroutine.

L'indirizzo della locazione della prima istruzione di una subroutine è noto come il suo *indirizzo iniziale* o *di partenza*. Questo dev'essere definito in ogni programma che chiama la subroutine. Se la subroutine ed il programma chiamante vengono assemblati contemporaneamente, l'indirizzo iniziale della subroutine può essere definito da un'etichetta alla sua prima istruzione. Se la subroutine ed il programma chiamante non sono assemblati insieme, l'indirizzo iniziale della subroutine dev'essere definito esplicitamente nell'istruzione chiamante.[2]

Nella Tab. 6.8 sono elencate le istruzioni dell'MC68020 per chiamare una subroutine e per fare ritorno da essa. Le istruzioni di salto ad una subroutine BSR (*Branch to SubRoutine*) e JSR (*Jump to SubRoutine*) sono quelle che ne eseguono la chiamata. In entrambi i casi, l'esecuzione fa sì che l'indirizzo di longword dell'istruzione che segue la chiamata venga salvato in cima allo stack di sistema. Dopodiché, l'esecuzione proseguirà dall'indirizzo iniziale della subroutine. Nessun'altra informazione viene salvata dalla chiamata, per cui è il programmatore ad avere la responsabilità di preservare qualsiasi contenuto di registro — compreso il registro dei codici di condizione — che possa essere modificato dalla subroutine. Tali contenuti possono essere salvati prima della chiamata e ripristinati dopo il ritorno, ma una subroutine ben progettata salverà i valori e li ripristinerà prima di far ritorno al programma chiamante. Quest'ultimo metodo è più ragionevole, poiché in genere avvengono più chiamate ad una medesima subroutine. Inoltre, se le subroutine sono progettate separatamente, il programmatore che progetta il programma chiamante potrebbe non essere consapevole dei registri che vengono modificati dalla subroutine, a meno che non sia disponibile una buona documentazione. Pertanto, negli esempi che seguiranno saranno mostrate delle subroutine la cui esecuzione è *trasparente* al programma chiamante, tranne per la modifica di registri impiegati per riportare i valori calcolati dalla subroutine.

Sono disponibili due istruzioni di ritorno per l'MC68020. L'istruzione RTR (*ReTurn and Restore:* torna e ripristina) è utilizzata quando i codici di condizione contenuti nel relativo registro (CCR) sono stati salvati nello stack di sistema dalla subroutine. Altrimenti, viene usata RTS (*ReTurn from Subroutine:* torna da subroutine) che si limita a caricare nel contatore di programma l'indirizzo di ritorno prelevato dallo stack.

In questo paragrafo sono discusse le istruzioni per la chiamata e il ritorno e il loro impiego con semplici strutture di subroutine. Un esame più dettagliato delle subroutine sarà svolto nel cap. 9, in cui saranno discussi vari metodi per il passaggio di dati tra i programmi chiamanti e le subroutine. Nella presente discussione, si suppone che i parametri siano passati in registri del processore. L'impiego dello stack di sistema durante le chiamate di subroutine è stato già discusso nel cap. 4.

[2] Se una subroutine ha diversi punti di entrata, allora dev'essere definito ciascun indirizzo. Inoltre, quando programmi indipendenti vengono assemblati separatamente, i riferimenti esterni o "globali" sono di solito definiti quando i moduli sono collegati insieme.

Tab. 6.8 *Istruzioni per l'impiego di subroutine.*

Istruzione	Sintassi	Operazione	Commenti
Salto (*branch*) a subroutine	BSR <spost>	1. (SP) ← (SP) – 4; ((SP))← (PC) 2. (PC) ←(PC) + <spost>	<spost> è un intero con segno di 8 bit, di 16 bit o di 32 bit.
Salto (*jump*) a subroutine	JSR <EA>	1. (SP) ← (SP) – 4; ((SP))← (PC) 2. (PC) < (EA)	<EA> è un tipo d'indirizzamento di controllo.
Ritorno e ripristino dei codici di condizione	RTR	1. (CCR) ← (SP)[7:0]; (SP) ← (SP) + 2 2. (PC) ←(SP); (SP) + 4	(CCR) = (SR)[7:0]
Ritorno da subroutine	RTS	(PC) ←((SP)); (SP) ← (SP) + 4	—

Note:

1. SP denota il puntatore di stack di sistema.
2. CCR è il registro dei codici di condizione, cioè (SR)[7:0].
3. PC è il contatore di programma.

6.3.1 Chiamate di subroutine

Le istruzioni BSR e JSR causano un trasferimento del controllo all'indirizzo iniziale di una subroutine. Nell'istruzione di salto ad una subroutine:

```
BSR        <etichetta>
```

l'operando <etichetta> fa sì che l'assemblatore calcoli lo spostamento tra l'istruzione BSR e l'istruzione identificata da <etichetta>. Tale spostamento viene aggiunto al contenuto corrente del contatore di programma (la locazione BSR + 2) per calcolare la locazione iniziale della subroutine. Lo spostamento è memorizzato come un intero nella notazione in complemento a 2. L'istruzione BSR opera nella stessa maniera dell'istruzione BRA discussa nel sottopar. 6.2.1, tranne che l'indirizzo dell'istruzione che segue la BSR viene salvato sullo stack di sistema. Similmente, l'istruzione JSR di salto ad una subroutine è identica all'istruzione JMP, a parte il salvataggio dell'indirizzo di ritorno. L'intervallo d'indirizzamento dell'istruzione JSR è illimitato e può essere impiegata qualsiasi modalità d'indirizzamento di controllo. Quindi, per specificare l'indirizzo iniziale, sono ammessi i modi d'indirizzamento assoluto, relativo, e indiretto, tranne quello con postincremento e con predecremento. Per esempio, l'istruzione

```
JSR        4(A5)
```

impiega l'indirizzamento indiretto con spostamento ed effettua un salto all'istruzione situata quattro byte oltre l'indirizzo in A5. Il ritorno all'istruzione che segue la BSR o la JSR che chiama la subroutine è conseguito mediante l'esecuzione di una istruzione RTR o RTS nella subroutine.

Esempio 6-8

La Fig. 6.12 illustra la struttura generale della chiamata di una subroutine e mostra una "traccia" del risultato. La chiamata decrementa di 4 il valore iniziale (SP) = $4000 e salva (PC) alla locazione $3FFC. Nella subroutine, la prima espressione salva il contenuto del registro dei codici di condizione sullo stack alla locazione $3FFA, appena al di sotto (nella memoria) delle due word che contengono l'indirizzo di ritorno. L'istruzione MOVEM.L inserisce sullo stack tutti i 32 bit di ciascun registro specificato dalla lista, usando l'indirizzamento con predecremento. Sono utilizzati 60 byte. Dopo l'elaborazione, i registri vengono ripristinati dall'ultima istruzione MOVEM prima del ritorno. L'istruzione RTR ripristina i codici di condizione nel registro di stato, lasciando invariato il byte superiore di SR prima del ritorno. Se i codici di condizione non fossero stati salvati all'atto dell'entrata nella subroutine, il ritorno avverrebbe mediante l'istruzione RTS. Questa istruzione si limita a caricare nel contatore di programma l'indirizzo di ritorno prelevato dallo stack.

```
                                 1.        TTL     FIGURA 6.12
                                 2.        LLEN    100
00010000                         3.        ORG     $10000
                                 4.  *
                                 5.  *   PROGRAMMA PRINCIPALE
                                 6.  *      ...
00010000  4EB9 0001000A          7.        JSR     SUBR
                                 8.  *      ...
00010006  4E4F                   9.        TRAP    #15                 ;RITORNA AL MONITOR
00010008  0063                  10.        DC.W    $0063               ; AL TERMINE DEL PROGRAMMA
                                11.  *                                 ; PRINCIPALE
                                12.  *
                                13.  *   SUBROUTINE
                                14.  *
0001000A  42E7                  15.  SUBR  MOVE.W  CCR,-(SP)           ;SALVA CODICI DI CONDIZIONE
0001000C  48E7 FFFE             16.        MOVEM.L D0-D7/A0-A6,-(SP)   ;SALVA I REGISTRI
                                17.  *      ...
00010010  4E71                  18.        NOP
                                19.  *      ...
00010012  4CDF 7FFF             20.        MOVEM.L (SP)+,D0-D7/A0-A6   ;RIPRISTINA I REGISTRI
                                21.  *
                                22.  *   RITORNA E RIPRISTINA I CODICI DI CONDIZIONE
                                23.  *
00010016  4E77                  24.        RTR
00010018                        25.        END
```

```
133Bug>RD
PC   =00010000 SR   =2700=TR:OFF_S._7_.....
USP  =00003830 MSP  =00003C18 ISP* =00004000 VBR  =00000000
SFC  =0=XX     DFC  =0=XX     CACR =0=..     CAAR =00000000
D0   =00000000 D1   =11111111 D2   =22222222 D3   =33333333
D4   =44444444 D5   =55555555 D6   =66666666 D7   =77777777
A0   =00000000 A1   =AAAA1111 A2   =AAAA2222 A3   =AAAA3333
A4   =AAAA4444 A5   =AAAA5555 A6   =AAAA6666 A7   =00004000
00010000 4EB90001 000A       JSR        ($1000A).L
133Bug>GT 10010
Effective address: 00010010
Effective address: 00010000
At Breakpoint
PC   =00010010 SR   =2700=TR:OFF_S._7_.....
USP  =00003830 MSP  =00003C18 ISP* =00003FBE VBR  =00000000
SFC  =0=XX     DFC  =0=XX     CACR =0=..     CAAR =00000000
D0   =00000000 D1   =11111111 D2   =22222222 D3   =33333333
D4   =44444444 D5   =55555555 D6   =66666666 D7   =77777777
A0   =00000000 A1   =AAAA1111 A2   =AAAA2222 A3   =AAAA3333
A4   =AAAA4444 A5   =AAAA5555 A6   =AAAA6666 A7   =00003FBE
00010010 4E71                NOP
133Bug>MD 3FBE:42
00003FBE 0000  0000  1111  1111   2222  2222  3333  3333    ........""""3333
00003FCE 4444  4444  5555  5555   6666  6666  7777  7777    DDDDUUUUffffwwww
00003FDE 0000  0000  AAAA  1111   AAAA  2222  AAAA  3333    ....**..**""**33
00003FEE AAAA  4444  AAAA  5555   AAAA  6666  0000  0001    **DD**UU**ff....
00003FFE 0006  0000  0000  FFFF   FFFF  0000  0000  FFFF    ................
0000400E FFFF  0000  0000  FFFF   FFFF  0000  0000  FFFF    ................
0000401E FFFF  0000  0000  FFFF   FFFF  0000  0000  FFFF    ................
0000402E FFFF  0000  0000  FFFF   FFFF  0000  0000  FFFF    ................
0000403E FFFF  0000                                         ....
```

Fig. 6.12 *(a) Struttura di modulo di subroutine. (b) traccia di programma dell'esempio 6-8.*

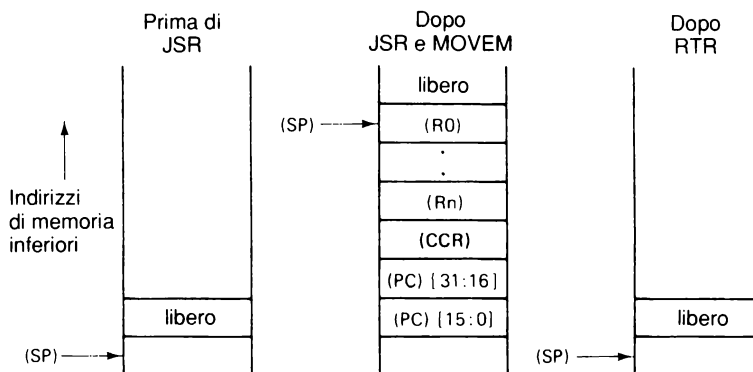
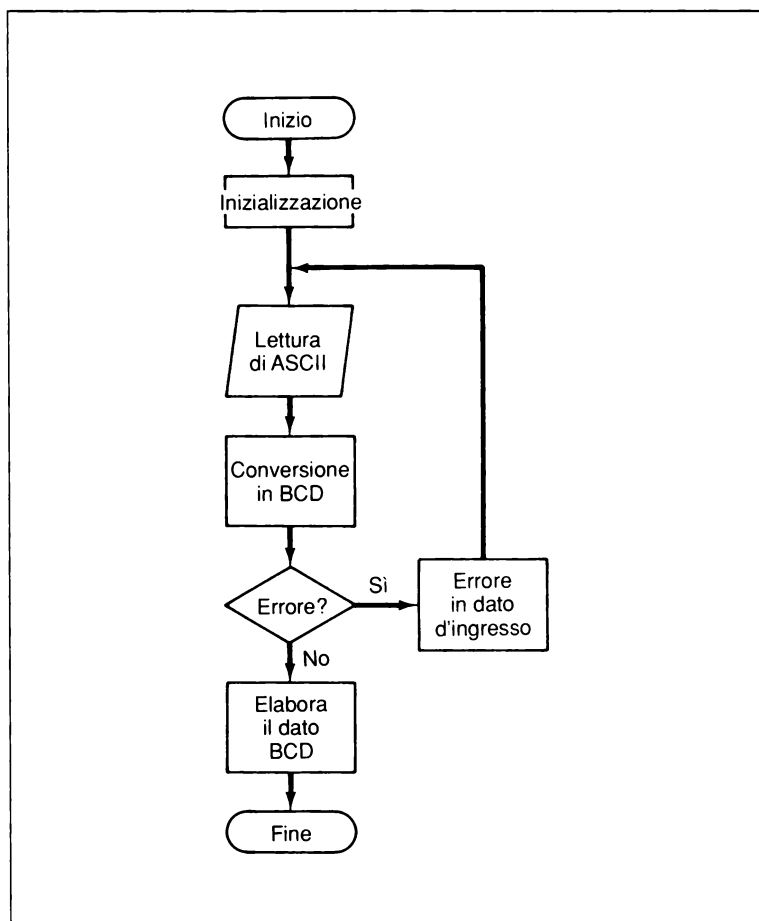
La Fig. 6.13 mostra la forma generale del contenuto dello stack, prima, durante e dopo la chiamata a SUBR. Il (PC) occupa sempre due word nello stack. Il contenuto di 8 bit del registro dei codici di condizione viene salvato nella word successiva (inferiore) della memoria. Dopodiché viene salvato ogni registro quando l'istruzione MOVEM viene eseguita. Il puntatore dello stack di sistema punta ora alla nuova cima dello stack, e lo stack potrà essere usato da routine d'interruzione o durante chiamate ad altre subroutine. Dopo che la subroutine ha completato la sua elaborazione, l'istruzione di ritorno dovrebbe lasciare il puntatore di stack al suo valore originale.

Nota: Ri indica un registro d'indirizzo o un registro di dati.

Fig. 6.13 *Utilizzazione dello stack di sistema durante una chiamata di subroutine.*

6.3.2 Struttura del programma

I grandi programmi possono essere suddivisi in un certo numero di subroutine per svolgere compiti specifici. Questo tipo di strutturazione del programma ha il vantaggio di semplificare il test del programma e di migliorarne la gestione. Il tempo di esecuzione è maggiore di quello impiegato da un programma simile senza subroutine, a causa del tempo richiesto dalla sequenza di chiamata e ritorno per ciascuna subroutine. Nella maggior parte dei casi, il requisito di modularità è più importante della riduzione del tempo di esecuzione.

Nella Fig. 6.14 è mostrata una possibile struttura di un programma per leggere i caratteri da una tastiera, convertirli in BCD e quindi elaborare i valori numerici. Il programma comincia inizializzando i valori, poi chiama la prima subroutine per leggere o acquisire i valori man mano che vengono digitati. Dopo che una stringa

Fig. 6.14
Esempio di struttura di programma.

di caratteri è stata acquisita, ognuno di essi viene esaminato da una seconda subroutine. Questo test consiste nell'esaminare ciascun carattere per verificarne la validità come carattere numerico ASCII; dopodiché, i caratteri validi saranno convertiti in BCD. Quando viene rivelato un carattere non valido, viene chiamata una subroutine di errore per visualizzare un messaggio di dato non valido sullo schermo CRT. Se non viene rivelato alcun errore, i dati BCD potranno essere elaborati ulteriormente da altre subroutine.

Nel prossimo capitolo, saranno trattate alcune tecniche per l'aritmetica binaria e BCD. Il paragrafo finale del cap. 7 presenterà le routine d'ingresso/uscita (*Input/Output:* I/O) e quelle di conversione, che consentiranno ad un programmatore di creare moduli di programma completi, come quello illustrato a grandi linee nella Fig. 6.14.

ESERCIZI

6.3.1

Se l'istruzione

```
BSR     SUB1
```

è situata all'indirizzo $3012 e l'etichetta SUB1 è in $3022, si definisca l'istruzione di linguaggio-macchina per

```
BSR     SUB1
```

Si consulti l'app. D per il formato di linguaggio-macchina.

6.3.2

Si disegni un diagramma dello stack per il modulo di programma nell'esempio 6.8 se (SP) = $7FFE inizialmente.

6.3.3

Nell'MC68020, qual è il fattore che limita il numero di subroutine che possono essere annidate? Una subroutine è definita "annidata" quando viene chiamata da un'altra subroutine.

6.3.4

Si modifichi il programma dell'esempio 6.3 per usare le chiamate di subroutine (JSR) anziché le istruzioni JMP.

CAPITOLO 7

OPERAZIONI ARITMETICHE

Questo capitolo concerne le operazioni aritmetiche su interi. S'impiegano dei dati numerici per identificare locazioni tramite i loro indirizzi e per rappresentare grandezze quali la temperatura o la pressione. In ogni caso, le operazioni fondamentali di addizione, sottrazione, moltiplicazione e divisione di numeri sono essenziali in algoritmi progettati per elaborare dati numerici. Quasi tutti i programmi incorporano qualche forma di operazione aritmetica.

Il sistema numerico impiegato nei microprocessori è quello binario. L'uso interno dell'aritmetica binaria e l'interpretazione esterna dei risultati come numeri decimali non causano alcun problema se sono disponibili le opportune routine di conversione. I valori decimali possono essere convertiti in codici ASCII per l'ingresso al sistema del computer e poi convertiti in binario per l'elaborazione. Per l'uscita possono essere effettuate le conversioni inverse. Tuttavia, in certi casi sorge un problema, dovuto alla lunghezza finita della rappresentazione di macchina. In altri termini, la lunghezza in bit di un valore intero è una misura del massimo intero rappresentabile in un certo formato. Quindi una rappresentazione di 32 bit limita la grandezza di un intero positivo a $2^{32} - 1$. Ogni operazione, come l'addizione di due numeri di 32 bit, presenta il rischio di superare il massimo valore consentito. I test per verificare tale condizione ed i mezzi per estendere la precisione quando è necessario sono due aspetti importanti dello studio delle operazioni aritmetiche svolte con l'ausilio del computer.

Si consiglia di rivedere gli argomenti del cap. 3 prima o durante lo studio di questo capitolo. Molti dettagli matematici già presentati allora non saranno ripetuti qui. Nell'app. D sono riassunti i valori assunti dai codici di condizione in seguito dall'esecuzione delle istruzioni dell'MC68020. L'artimetica in virgola mobile sarà presentata nel cap. 12.

Nei primi paragrafi di questo capitolo, saranno presentate le istruzioni aritmetiche fondamentali dell'MC68020. Esse comprendono ADD, SUB, EXT, NEG, MUL e DIV per operandi di 8, 16 o 32 bit trattati come interi binari. Quando la lunghezza degli operandi o del risultato supera i 32 bit, sono impiegati metodi di precisione multipla, che saranno discussi nel par. 7.4. Le istruzioni di aritmetica decimale

in singola e multipla precisione, tra cui ABCD, DBCD e NBCD, saranno trattate nel par. 7.5. Il paragrafo finale tratterà l'importante argomento delle tecniche d'ingresso/uscita che utilizzano il programma supervisore (monitor). Tale discussione comprenderà i metodi di conversione tra caratteri ASCII e valori binari o decimali.

7.1 ALCUNI DETTAGLI DELL'ARITMETICA BINARIA

Nell'MC68020, la rappresentazione di operandi binari in singola precisione può avere una lunghezza di 8, 16 o 32 bit. Numericamente i valori interi positivi si estendono da 0 a $2^m - 1$, dove $m = 8$, 16 o 32. Per gli interi con segno, l'intervallo è da -2^{m-1} a $+2^{m-1} - 1$ nella notazione in complemento a 2. Nella somma o nella sottrazione di interi di m bit, possono presentarsi varie condizioni di fuori-intervallo, che sono indicate dai bit dei codici di condizione del processore. Il caso di interi senza segno è trattato diversamente da quello degli interi con segno, poiché cambia l'interpretazione dei codici di condizione.

Si consideri l'addizione di due interi di m cifre senza segno. Per esempio, l'addizione con 8 bit di 125 e 200 fornisce:

```
      Binario
     0111  1101
   + 1100  1000
     ----  ----
 (1) 0100  0101
```

cioè un risultato 69 di 8 bit con un riporto. Questa addizione nell'MC68020 causerebbe l'attivazione del codice di condizione di riporto (*carry*), C = {1}, per indicare appunto che è avvenuto un riporto. Allo stesso modo, una sottrazione in cui il sottraendo è maggiore del minuendo è indicata da un riporto negativo (*borrow*) dalla ($m + 1$)-ma cifra. Nell'MC68020, anche il riporto negativo è indicato dalla condizione C = {1}. Quindi, ogni volta che il bit di riporto è {1} dopo un'addizione o una sottrazione di interi senza segno, il risultato di m bit non è corretto, per cui il programmatore dovrà decidere in merito all'azione da intraprendere.

Quando sono sommati o sottratti numeri interi nella notazione in complemento a 2, il bit di riporto viene ignorato, mentre il bit di overflow (V) viene esaminato per rivelare un'eventuale condizione di fuori-intervallo. Per una rappresentazione con m bit, l'intervallo positivo si estende soltanto da 0 a $2^{m-1} - 1$, per cui l'addizione di due numeri positivi non deve superare quel valore. Per esempio, l'addizione di 8 bit di +125 e +127 fornisce:

```
      Binario
     0111  1101
   + 0111  1111
     ----  ----
     1111  1100
```

cioè –4 (un risultato chiaramente errato). Anche la condizione di overflow, V = {1}, è indicata. Il problema è un risultato negativo che è stato ottenuto dalla somma di due numeri positivi. Similmente, se l'addizione di numeri negativi producesse un risultato positivo, il codice di condizione V sarebbe nuovamente posto a {1}. Questa condizione matematica è definita *underflow* e significa che il risultato è troppo piccolo (troppo negativo) per poter essere rappresentato. Comunque, il bit V di overflow indica l'errore, ed il riferimento a condizioni di fuori intervallo nella macchina è di solito denotato come "overflow". Se gli operandi sono di segno opposto, non può presentarsi alcuna condizione di fuori-intervallo. Quando viene eseguita una sottrazione, il bit V è di nuovo posto a {1} per indicare un risultato errato.

Nell'MC68020, anche la divisione e l'aritmetica BCD possono produrre condizioni di fuori-intervallo. Le condizioni per questi casi saranno trattate nei paragrafi appropriati di questo capitolo. In sostanza, i codici di condizione devono essere esaminati dopo un'operazione aritmetica, per determinare se si è verificata una condizione di fuori-intervallo. La Fig. 7.1 indica la procedura adottata per verificare la validità dei numeri. Una condizione di errore indica che la rappresentazione con *m* bit non è valida: o il risultato viene rifiutato o è richiesta una rappresentazione di precisione multipla.

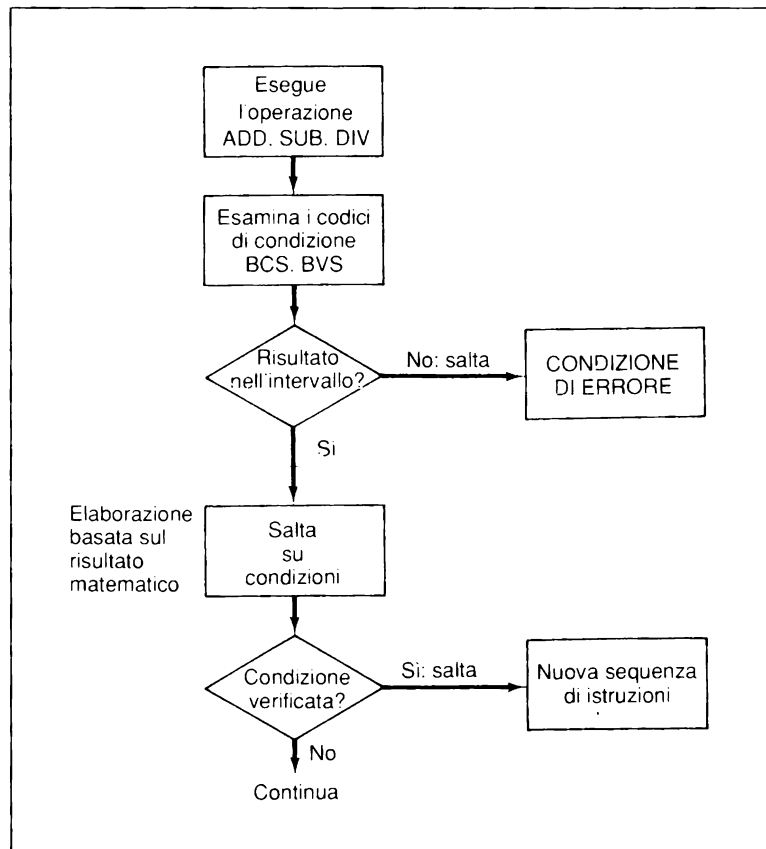



Fig. 7.1
Test per le condizioni di fuori-intervallo.

Test condizionali. Un'istruzione BCS (*Branch if Carry Set:* salta se riporto avvenuto) posta dopo un'istruzione aritmetica produce un salto se è stato generato un riporto durante un'addizione o un riporto negativo durante una sottrazione di interi senza segno. BVS (*Branch if oVerflow Set:* salta se overflow avvenuto) causa un salto quando si verifica una condizione di fuori-intervallo durante l'aritmetica con interi in complemento a 2. Si consiglia d'impiegare tali test (o quelli logicamente opposti di BCC e BVC) prima che siano eseguiti altri test condizionali.

Quando vengono eseguite istruzioni di salto condizionato dopo istruzioni aritmetiche, i test condizionali descritti nel cap. 6 devono essere usati con cura. Si consideri l'addizione appena mostrata di +125 e +127 per numeri interi di 8 bit con segno. I codici di condizione dopo l'istruzione ADD sarebbero:

$Z = \{0\}$, $N = \{1\}$, $C = \{0\}$, $V = \{1\}$

che indicano, rispettivamente: non zero, negativo, nessun riporto, overflow. Il risultato è matematicamente errato, ma l'istruzione BMI causerebbe un salto, poiché essa esamina soltanto la condizione $N = \{1\}$. Peggio ancora, BGE produrrebbe un salto, ma BPL no. BGE causa un salto ogni volta che i codici di condizione N e V coincidono, ma BPL salta quando $N = \{0\}$, senza esaminare alcun altro codice di condizione.

Se non è presente alcuna condizione di fuori-intervallo, i test condizionali agiscono come mostrato nella Fig. 7.2(a). In questo caso, BPL e BGE sono equivalenti per interi con segno validi, e così pure BMI e BLT. La Fig. 7.2(b) illustra le condizioni e i test sui codici di condizione. La "codifica" definisce la condizione nell'istruzione di linguaggio-macchina. La sequenza di istruzioni nella Fig. 7.1 è seguita nella maggior parte degli esempi in questo capitolo in cui può presentarsi una condizione di fuori-intervallo.

Istruzione	Condizione di salto	Risultato
SENZA SEGNO		
ADD, SUB $C = \{0\}$	BEQ BNE	$X = 0$ $X \neq 0$
CON SEGNO		
ADD, SUB $V = \{0\}$	BEQ BNE BPL, BGE BLE BMI, BLT BGT	$X = 0$ $X \neq 0$ $X \geq 0$ $X \leq 0$ $X < 0$ $X > 0$

Fig. 7.2 *(a) Test senza segno e con segno.*

Mnemonico	Condizione	Codifica	Test
T	True (vero)	0000	1
F	*False* (falso)	0001	0
HI	*HIgh* (alto)	0010	$\overline{C} \cdot \overline{Z}$
LS	*Low or Same* (basso o identico)	0011	$C + Z$
CC (HS)	*Carry Clear* (non riporto)	0100	$\overline{C}$
CS (LO)	*Carry Set* (riporto)	0101	C
NE	*Not Equal* (non uguale)	0110	$\overline{Z}$
EQ	*Equal* (uguale)	0111	Z
VC	*oVerflow Clear* (non overflow)	1000	$\overline{V}$
VS	*oVerflow Set* (overflow)	1001	V
PL	*PLus* (più)	1010	$\overline{N}$
MI	*MInus* (meno)	1011	N
GE	*Greater or Equal* (maggiore o uguale)	1100	$N \cdot V + \overline{N} \cdot \overline{V}$
LT	*Less Than* (minore)	1101	$N \cdot \overline{V} + \overline{N} \cdot V$
GT	*Greater Than* (maggiore)	1110	$N \cdot V \cdot \overline{Z} + \overline{N} \cdot \overline{V} \cdot \overline{Z}$
LE	*Less or Equal* (minore o uguale)	1111	$Z + N \cdot \overline{V} + \overline{N} \cdot V$

Figura 7.2 *(b) Test dei codici di condizione.*

ESERCIZI

7.1.1 Si confronti l'impiego delle istruzioni Bcc dopo operazioni aritmetiche col loro impiego dopo istruzioni MOVE, CMP e TST, discusso nel cap. 6.

7.1.2 Si mostri che BGE e BPL forniscono risultati opposti quando si verifica un overflow.

7.1.3 Si determini il risultato della sottrazione di –32768 da 16384 usando l'aritmetica in complemento a 2 di 16 bit.

7.1.4 Si dimostri che è possibile effettuare la sottrazione di numeri binari sommando al minuendo il complemento a 2 del sottraendo.

7.2 ADDIZIONE E SOTTRAZIONE

L'addizione binaria è eseguita su operandi di 8, 16 o 32 bit mediante l'istruzione ADD e le sue varianti, come mostrato nella Tab. 7.1. Similmente, gli operandi di byte, di word o di longword possono essere sottratti con le istruzioni SUB o le sue varianti. Le istruzioni EXT e EXTB estenderanno di segno un operando da una lunghezza all'altra. Queste istruzioni sono utili quando s'incontrano operandi di diverse lunghezze in un'applicazione.

Tab. 7.1 *Le istruzioni aritmetiche ADD, SUB, EXT e NEG.*

Sintassi		Modalità d'indirizzamento	
		Sorgente	**Destinazione**
Addizione o sottrazione			
ADD.<l>	<EA>,<Dn>	Tutte[1]	Dn
SUB.<l>	<EA>,<Dn>		
ADD.<l>	<Dn>,<EA>	Dn	Alterabile di memoria
SUB.<l>	<Dn>,<EA>		
ADDI.<l>	#<d>,<EA>	#<d>	Alterabile di dati
SUBI.<l>	#<d>,<EA>		
ADDQ.<l>	#<d3>,<EA>	#<d3>[2]	Alterabile[3]
SUBQ.<l>	#<d3>,<EA>		
Estensione di segno			
EXT.W	<Dn>	(Dn)[7:0]	(Dn)[15:0]
EXT.L	<Dn>	(Dn)[15:0]	(Dn)[15:0]
EXTB.L	<Dn>	(Dn)[7:0]	(Dn)[15:0]
Negazione			
NEG.<l>	<EA>	---	Alterabile di dati

L'istruzione NEG forma il complemento a 2 (negativo) dell'operando specificato. Queste istruzioni e le loro varianti sono usate per eseguire le operazioni aritmetiche fondamentali su numeri binari. Come si può rilevare dalla Tab. 7.1, ciascuna istruzione ha delle limitazioni per le modalità d'indirizzamento consentite. Nell'operazione di ciascuna istruzione, definita nella Tab. 7.2, l'operando di destinazione di lunghezza specificata è sostituito dal risultato. Questo è la somma per ADD, la differenza per SUB, o il valore negativo quando viene eseguita NEG. I valori dei codici di condizione N, Z, V e C sono assegnati in base al risultato. Anche il codice di condizione X (*eXtend:* estensione) viene fissato per l'impiego in operazioni di precisione multipla.

Tab. 7.2 *Operazioni delle istruzioni aritmetiche.*

Istruzione		**Operazione**
ADD.<l>	<EAs>,<EAd>	(EAd)[l] ← (EAs)[l] + (EAd)[l]
SUB.<l>	<EAs>,<EAd>	(EAd)[l] ← (EAd)[l] – (EAs)[l]
EXT.W	<Dn>	(Dn)[W] ← (Dn)[B]
EXT.L	<Dn>	(Dn)[L] ← (Dn)[W]
EXTB.L	<Dn>	(Dn)[L] ← (Dn)[B]
NEG.<l>	<EA>	(EA)[l] ← 0 – (EA)[l]

Note:
1. <EAs> e <EAd> denotano gli indirizzi effettivi di sorgente e di destinazione, rispettivamente.
2. <l> denota B, W o L.
3. [l] indica i corrispondenti bit nell'operazione.
4. Le istruzioni EXT e EXTB estendono di segno l'operando di sorgente nel medesimo registro.

Note alla Tab. 7.1:

1. Se l'indirizzo effettivo di sorgente nelle istruzioni ADD o SUB è in un registro d'indirizzo, allora la lunghezza di operando è una word o una longword.
2. <d$_3$> è un valore da 1 a 8.
3. Se An è una destinazione, sono ammesse soltanto operazioni di word o di longword. In questo caso, i codici di condizione non sono interessati.
4. <l> denota B, W o L in tutte le istruzioni tranne quelle nelle note 1 e 3.
5. Tranne che nella nota 3, tutti i codici di condizione sono interessati dalle istruzioni aritmetiche. EXT e EXTB definiscono N e Z in base al risultato, ma azzerano V e C.

Addizione. Il formato in linguaggio assembler dell'istruzione ADD è:

```
ADD.<l>    <EA>,<Dn>
```

dove l'operando di destinazione è contenuto in un registro di dati. La sorgente è specificata come un operando di byte, di word o di longword (<l> = B, W o L), a meno che la sorgente non sia un registro d'indirizzo, nel qual caso la lunghezza è limitata ad operandi di word o di longword. Quindi l'istruzione

```
ADD.B    D1,D5
```

sostituisce (D5)[7:0] con la somma (D5)[7:0] + (D1)[7:0]. Soltanto la lunghezza specificata della destinazione è interessata. Come mostrato nella Tab. 7.1, un registro di dati potrebbe contenere l'operando di sorgente quando la destinazione <EA> è definita da una modalità d'indirizzamento alterabile di memoria.

È ammessa anche la somma del valore in un registro di dati ad un operando nella memoria. Per esempio, l'istruzione

```
ADD.L    D1,(A1)
```

somma il contenuto di 32 bit di D1 al contenuto della locazione indirizzata da A1. Tuttavia, l'operando di destinazione non può essere il contenuto di un registro d'indirizzo, né può essere indirizzato dal modo d'indirizzamento relativo al contatore di programma.

Alcune varianti dell'istruzione ADD mostrate nella Fig. 7.2 sono ADDI (*ADD Immediate:* somma immediata) e ADDQ (*ADD Quick:* somma rapida). Il formato immediato può sommare una constante di 8, 16 o 32 bit ad una locazione alterabile di dati. Ciò esclude un registro d'indirizzo o un indirizzo relativo al PC per la destinazione. L'istruzione ADDI somma una constante specificata ad una locazione di destinazione, mentre l'istruzione ADD opera soltanto tra registri e memoria. Pertanto, l'istruzione

```
ADDI.B    #20,(A1)
```

è usata per sommare la costante 20 al byte indirizzato da A1 in questo esempio.

L'istruzione ADDQ aggiunge un valore immediato compreso tra 1 e 8 alla locazione di destinazione specificata. È ammessa qualsiasi locazione alterabile come destinazione, compreso un registro d'indirizzo. Sono proibite destinazioni indirizzate dal modo relativo al PC. In quei casi in cui la destinazione è An, i codici di condizione non sono interessati, ma l'intero registro (32 bit) viene modificato. Le operazioni aritmetiche con indirizzi saranno considerate nel cap. 9.

Si dovrebbe notare che alcuni assemblatori selezionano la variante appropriata di un'istruzione ADD senza che il programmatore debba specificarla. Per esempio, l'istruzione

```
ADD    #1,D4
```

può essere interpretata come un'istruzione ADDI o un'istruzione ADDQ da certi assemblatori. La pratica di lasciare all'assemblatore il compito di scegliere la variante non sempre è opportuna, se si desidera ottenere una coerenza tra la documentazione del programma e la lunghezza delle istruzioni di linguaggio-macchina.

Sottrazione. Le istruzioni SUB, SUBI E SUBQ sono le controparti esatte delle istruzioni di addizione. Esse calcolano la differenza tra l'operando di destinazione (minuendo) e l'operando di sorgente (sottraendo). Questa differenza sostituisce la porzione di byte, di word o di longword della locazione di destinazione, come specificato dall'istruzione.

Se un registro d'indirizzo è la destinazione in un'istruzione SUBQ, il valore immediato può essere un operando di word o di longword. Se la lunghezza è di una word, allora il valore immediato viene esteso di segno a 32 bit prima che la sottrazione sia eseguita. Per esempio, l'istruzione

```
SUBQ.W    #1,A1
```

ha l'effetto di sottrarre 1 dall'intero contenuto di A1. Pertanto, sia ADDQ che SUBQ influiscono sul contenuto di 32 bit del registro d'indirizzo di destinazione.

Tutti i bit dei codici di condizione sono interessati da un'istruzione di sottrazione, a meno che tale operazione non sia effettuata da un registro d'indirizzo. La condizione C = {1} indica un riporto negativo in una sottrazione di operandi senza segno. Nell'aritmetica con segno, i bit N, Z e V posti a 1 indicano un numero negativo, o nullo, o una condizione di overflow, rispettivamente.

Estensione. Ogni istruzione di estensione di segno estende un intero con segno alla lunghezza specificata. Per esempio, l'istruzione

```
EXT.W     D1
```

estende (cioè, copia) il bit [7] del registro di dati designato D1 nei bit [15:8]. Le tre istruzioni di estensione del segno servono ad estendere un byte ad una word, una word ad una longword, o un byte ad una longword. L'aritmetica su variabili di lunghezza mista è comunemente impiegata quando dispositivi differenti generano numeri con segno. Per esempio, un convertitore analogico/digitale potrebbe emettere in uscita i suoi valori digitali come un intero in complemento a 2 di 16 bit. Se s'impiega l'aritmetica di longword per elaborare i valori, ciascun valore di 16 bit dev'essere esteso di segno a 32 bit dall'istruzione EXT.L.

Esempio 7-1

La subroutine mostrata nella Fig. 7.3 somma gli elementi di una colonna (elementi vettoriali) di interi di 16 bit per formare una somma di 16 o di 32 bit. Prima che la subroutine sia chiamata, D1 contiene il numero di interi da sommare, mentre A1 contiene l'indirizzo iniziale della colonna. Se la lunghezza

della colonna è zero, D3 conterrà –1 dopo l'esecuzione della subroutine. Altrimenti, la word meno significativa di D3 conterrà 0 se la somma è contenuta in 16 bit, o +1 se la somma richiede 32 bit. Ogni volta che viene generato un riporto nell'addizione di un nuovo valore a D2, un 1 viene aggiunto alla word superiore di D2 per formare la somma corretta.

Lo scopo del programma potrebbe essere quello d'impiegare operazioni di word (anziché di longword) nell'elaborazione successiva, se il risultato è una somma di 16 bit. Ciò ridurrebbe la quantità di memoria richiesta e sarebbe più efficiente in termini di tempo se sono effettuati accessi alla memoria.

```
                              1.          TTL     FIGURA 7.3
                              2.          LLEN    100
00020000                      3.          ORG     $20000
                              4.  *
                              5.  *   SOMMA INTERI CON SEGNO DI 16 BIT
                              6.  *
                              7.  *       INPUT:  (D1.W) = NUMERO DI INTERI
                              8.  *               (A1.L) = INDIRIZZO INIZIALE DI COLONNA
                              9.  *                        DI INTERI
                              10. *
                              11. *       OUTPUT: (D3.W) = -1 : IL NUMERO DI INTERI E' ZERO
                              12. *                         0 : SOMMA DI 16 BIT IN D2 [15:0]
                              13. *                         1 : SOMMA DI 32 BIT IN D2 [31:0]
                              14. *               (D2.L) = SOMMA
                              15. *
00020000  48E7 4040           16. ADDUNS  MOVEM.L D1/A1,-(SP)      ;SALVA REGISTRI
                              17. *
00020004  4282                18.         CLR.L   D2               ;SOMMA := 0
00020006  363C FFFF           19.         MOVE.W  #-1,D3           ;IMPOSTA STATO PER ERRORE
0002000A  5341                20.         SUBQ.W  #1,D1            ;PREDISPONE CONTATORE DI CICLO
0002000C  6D00 0016           21.         BLT     FINE
00020010  4243                22.         CLR.W   D3               ;IMPOSTA STATO PER SOMMA DI 16 BIT
                              23. *
00020012  D459                24. LOOP    ADD.W   (A1)+,D2
00020014  6400 000A           25.         BCC     ENDLP            ;SE NON C'E' OVERFLOW, SALTA
00020018  0682 00010000       26.         ADDI.L  #$10000,D2       ;ALTRIMENTI, SOMMA 1 ALLA WORD SUP.
0002001E  7601                27.         MOVEQ   #1,D3            ;IMPOSTA STATO PER SOMMA DI 32 BIT
00020020  51C9 FFF0           28. ENDLP   DBRA    D1,LOOP          ;CONTINUA
00020024  4CDF 0202           29. FINE    MOVEM.L (SP)+,D1/A1      ;RIPRISTINA REGISTRI
00020028  4E75                30.         RTS
0002002A                      31.         END
```

Fig. 7.3 *Routine di addizione per somme di 16 bit o di 32 bit.*

Negazione. L'istruzione NEG sostituisce una locazione di destinazione alterabile di dati, di lunghezza specificata, col risultato del seguente calcolo:

0 – (destinazione)

formando così il valore in complemento a 2. Come esempio, se la locazione $1000 contiene il valore –1, allora l'istruzione

```
NEG.B     $1000
```

sostituisce il valore con $FF. L'unico caso in cui può presentarsi una condizione di overflow è quando il valore negato è -2^{m-1}, poiché questo numero non ha un equivalente positivo nella notazione in complemento a 2.

Esempio 7-2

La subroutine mostrata nella Fig. 7.4 somma le differenze tra gli elementi di due colonne o vettori di interi indirizzati da (A1) e (A2). La lunghezza delle colonne è contenuta inizialmente in D1. Se la lunghezza delle colonne è zero o è avvenuto un overflow, allora i bit (D4)[15:0] vengono azzerati per indicare l'errore. Altrimenti, le differenze tra gli elementi corrispondenti delle colonne vengono accumulate nella word meno significativa di D3. Una volta che la somma è completa, il risultato viene esaminato per verificarne il segno. Se il risultato è negativo, l'istruzione NEG è usata per determinare la grandezza del numero. Un programma che impiega questa routine deve dapprima esaminare l'uscita di stato in D4 per verificare che non ci sia alcuna condizione di errore. Se nessun errore è indicato, allora la grandezza della somma delle differenze è contenuta nella word meno significativa di D3. Il programma potrebbe quindi convertire la grandezza in un numero decimale ASCII e anteporre il segno al risultato stampato. Se avviene un overflow con le operazioni aritmetiche di 16 bit nel programma, allora si potrebbe impiegare l'aritmetica di 32 bit.

```
                         1.          TTL      FIGURA 7.4
                         2.          LLEN     100
00020000                 3.          ORG      $20000
                         4.  *
                         5.  *    SOMMA DELLE DIFFERENZE DI DUE COLONNE
                         6.  *    DI INTERI DI 16 BIT
                         7.  *
                         8.  *       INPUT:  (D1.W) = LUNGHEZZA DELLE COLONNE
                         9.  *               (A1.L) = INDIRIZZO DELLA PRIMA COLONNA
                        10.  *               (A2.L) = INDIRIZZO DELLA SECONDA COLONNA
                        11.  *
                        12.  *       OUTPUT: (D3.W) = VALORE ASSOLUTO DEL RISULTATO
                        13.  *               (D4.W) =  0 : E' AVVENUTO UN ERRORE
                        14.  *                         1 : IL RISULTATO E' POSITIVO
                        15.  *                        -1 : IL RISULTATO E' NEGATIVO
                        16.  *
00020000  48E7 6060     17.  SUMDIF  MOVEM.L D1/D2/A1/A2,-(SP) ;SALVA I REGISTRI
00020004  4243          18.          CLR.W    D3                ;SOMMA := 0
00020006  4244          19.          CLR.W    D4                ;IMPOSTA STATO NORMALE PER ERRORE
00020008  4A41          20.          TST.W    D1                ;SE LE COLONNE SONO VUOTE,
0002000A  6700 0026     21.          BEQ      FINE              ;POI TORNA CON ERRORE
                        22.  *
0002000E  3419          23.  LOOP    MOVE.W   (A1)+,D2          ;CALCOLA
00020010  945A          24.          SUB.W    (A2)+,D2          ;((A1)) - ((A2))
00020012  6900 001E     25.          BVS      FINE              ;ESCE SE OVERFLOW
00020016  D642          26.          ADD.W    D2,D3             ;SOMMA LE DIFFERENZE
00020018  6900 0018     27.          BVS      FINE              ;SE OVERFLOW, ESCE CON ERRORE
0002001C  5341          28.          SUBQ.W   #1,D1             ;DECREMENTA IL CONTEGGIO
0002001E  66 EE         29.          BNE      LOOP              ;RIESEGUE IL CICLO FINO ALLA FINE
                        30.  *
00020020  4A43          31.          TST      D3                ;SE IL RISULTATO E' POSITIVO
00020022  6C00 000C     32.          BGE      POS               ;ALLORA LO ELABORA
00020026  383C FFFF     33.          MOVE.W   #-1,D4            ;ALTRIMENTI FISSA STATO A NEGATIVO
0002002A  4443          34.          NEG.W    D3                ;PRENDE IL VALORE ASSOLUTO
0002000C  6000 0004     35.          BRA      FINE
                        36.  *
00020030  7801          37.  POS     MOVEQ    #1,D4             ;FISSA STATO A POSITIVO
00020032  4CDF 0606     38.  FINE    MOVEM.L (SP)+,D1/D2/A1/A2 ;RIPRISTINA I REGISTRI
00020036  4E75          39.          RTS
00020038                40.          END
```

Fig. 7.4 *Routine per calcolare la somma di differenze.*

ESERCIZI

7.2.1 Si supponga che (D1) = \$0000 FFFF prima dell'esecuzione di ciascuna delle istruzioni elencate di seguito. Si determinino i risultati, inclusi i valori assegnati ai codici di condizione, dopo ogni istruzione.

```
(a) ADDI.B   #1,D1
(b) ADDQ.L   #1,D1
(c) SUBQ.B   #1,D1
(d) NEG.W    D1
(e) SUB.L    D1,D1
(f) EXT.L    D1
```

7.2.2 Si dimostri che l'impiego di un solo bit di riporto dopo un'addizione senza segno di *m* bit è sufficiente a garantire che non venga persa alcuna informazione.

7.2.3 Per un'operazione di sottrazione di *m* bit N3 = N2 – N1, si dimostri che il risultato corretto si ottiene quando N1 è negativo e N2 è positivo, se la somma delle grandezze di N1 e N2 è minore o uguale a $2^{m-1} - 1$. Si consulti il cap. 2 per le formule dell'aritmetica in complemento a 2.

7.2.4 Si scriva una routine per moltiplicare due interi senza segno di 16 bit usando l'addizione ripetuta.

7.2.5 Si modifichi il programma dell'esempio 7.2 per calcolare la somma dei valori assoluti delle differenze tra le due colonne di numeri.

7.3 MOLTIPLICAZIONE E DIVISIONE

Le istruzioni per la moltiplicazione e la divisione di interi sono incluse nell'insieme di istruzioni dei microprocessori a 32 bit. L'MC68020 dispone di istruzioni separate per la moltiplicazione e la divisione di interi senza segno e per la moltiplicazione e la divisione di interi in complemento a 2. In questo paragrafo sono presentate tali istruzioni e discusse le operazioni, gli intervalli numerici, e i possibili errori che possono sorgere nel loro impiego. La Tab. 7.3 illustra la sintassi delle operazioni dell'istruzione di divisione DIV e dell'istruzione di moltiplicazione MUL. Il suffisso "U" per gli interi senza segno o il suffisso "S" per gli interi con segno deve essere specificato ogni volta che essi sono usati. Il moltiplicando per l'operazione di moltiplicazione ed il divisore per l'operazione di divisione sono specificati da un operando di sorgente di 16 o di 32 bit. Soltanto l'indirizzamento diretto con registro d'indirizzo è proibito per la sorgente. Il moltiplicatore e il dividendo sono sempre contenuti in registri di dati. Il risultato è un valore di 32 bit contenuto in un registro di dati.

Tab. 7.3 *Istruzioni di moltiplicazione e di divisione.*

Sintassi		Operazione
Moltiplicazione		
MULU.W MULS.W	<EA>,<Dn>	(Dn)[31:0] ← (Dn)[15:0]*(EA)[15:0]
MULU.L MULS.L	<EA>,<Dn>	(Dn)[31:0] ← (Dn)[31:0]*(EA)[31:0]
Divisione		
DIVU.W DIVS.W	<EA>,<Dn>	(Dn)[31:0]/(EA)[15:0] (Dn)[15:0] ← quoziente (Dn)[31:16] ← resto
DIVU.L DIVS.L	<EA>,<Dn>	(Dn)[31:0]/(EA)[31:0] (Dn)[31:0] ← quoziente

Note:
1. Soltanto le modalità d'indirizzamento di dati sono ammesse per l'indirizzo di sorgente <EA> (cioè, An è proibito).
2. MULU.L e MULS.L causano la condizione V = {1} se il prodotto supera i 32 bit.
3. Nella divisione, un divisore zero causa una *trappola*; una condizione di overflow è indicata da V = {1}.
4. Nella divisione con segno, il resto ha il segno del dividendo.

Moltiplicazione senza segno. L'istruzione MULU.W moltiplica due operandi di 16 bit senza segno per fornire un prodotto di 32 bit. Per esempio, l'istruzione

```
MULU.W    $10,D2
```

con (D2)[15:0] = \$0002 dà come risultato (D2) = \$0000 0020, cioè 32 decimale. Poiché sia il moltiplicando che il moltiplicatore di 32 bit possono estendersi nell'intervallo da 0 a 65535, il prodotto non può superare 4294836225, che è minore di $2^{32} - 1$. Pertanto, non è possibile alcuna condizione di overflow e i bit dei codici di condizione C e V sono sempre azzerati dopo l'istruzione MULU.W. Invece N e Z sono impostati in base al risultato. Nel caso di interi senza segno, N = {1} indica che il prodotto ha una grandezza maggiore o uguale a 2^{31}.

È disponibile un'altra istruzione di moltiplicazione senza segno (MULU.L) per moltiplicare operandi di 32 bit. Tuttavia, il prodotto è limitato dal troncamento a 32 bit. Se il valore assoluto del prodotto supera $2^{32} - 1$, viene generata una condizione di overflow (V = {1}). Pertanto, questa istruzione è d'impiego limitato se le operazioni aritmetiche creano un valore che richiede più di 32 bit. In questo caso,

dev'essere impiegata l'aritmetica in precisione multipla. Comunque, l'istruzione è utile quando un prodotto non può superare 32 bit, come nel calcolo di un valore di indice, ad esempio. In ogni caso, le due istruzioni di moltiplicazione in precisione multipla forniscono al programmatore la flessibilità di scegliere la lunghezza dell'operando che è più adatta per un'applicazione specifica.

Moltiplicazione con segno. Quando vengono moltiplicati interi con segno, il risultato è positivo o negativo a seconda dei segni del moltiplicando e del moltiplicatore. L'intervallo di ciascuno di questi si estende da -2^{15} a $+2^{15} - 1$, cioè da –32768 a +32767. Se sono moltiplicati i due valori più negativi, il risultato è 1073741824, cioè 2^{30}. Il massimo risultato positivo è allora:

$$-2^{15} \times (2^{15} - 1) = -1073709056$$

Pertanto, non può presentarsi alcuna condizione di fuori- intervallo per un prodotto di 32 bit, e sia V che C sono sempre azzerati. Per MULS.W il bit N indica un prodotto negativo quando N = {1}, come previsto. Se il risultato è zero, allora Z = {1}. L'istruzione

```
MULS.W     #-1,D2
```

con (D2)[15:0] = $0002 risulta in (D2) = $FFFF FFFE, cioè –2 nella notazione in complemento a 2. Il bit N viene posto a {1} per indicare un risultato negativo.

L'istruzione MULS.L tronca a 32 bit il prodotto di due operandi di 32 bit, ponendo V = {1} se si verifica un overflow. Come nell'istruzione MULU.L, il bit del codice di condizione V dovrebbe essere verificato dopo l'esecuzione di questa istruzione, a meno che il prodotto non superi i 32 bit.

Divisione senza segno. L'istruzione DIVU.W dell'MC68020 esegue la divisione:

$$Y/W = Q + R/W$$

dove Y è un intero senza segno di 32 bit, W è un intero senza segno di 16 bit, Q è il quoziente di 16 bit, e R è il resto di 16 bit. Per esempio, l'istruzione

```
DIVU.W     #2,D1
```

divide per 2 l'operando di 32 bit contenuto in D1. Il risultato, come indicato dalla Tab. 7.3, è un quoziente nella word meno significativa di D1, mentre il resto (zero) è contenuto nella word più significativa di D1. Quindi, se D1 conteneva $0000 0005 prima che l'istruzione fosse eseguita, il risultato sarà (D1) = $0001 0002, poiché 5/2 = 2 + 1/2.

Possono presentarsi due condizioni speciali quando si esegue un'operazione di divisione:

(a) divisione per zero

(b) overflow del quoziente

Ognuna di queste situazioni è indicata da una condizione di errore. L'elaborazione dell'eccezione in una routine di trappola avviene nel caso di divisione per zero. Si può verificare un overflow poiché l'intervallo del dividendo è da 0 a $2^{32} - 1$, mentre la lunghezza del quoziente è soltanto di 16 bit. Ovviamente, se si dividesse per 1 un intero maggiore di $2^{16} - 1$, sarebbe generato un overflow, Più in generale, se il dividendo supera il divisore di almeno 2^{16} in valore assoluto, allora si presenterà una condizione di overflow. Tale condizione è indicata da V = {1} anche se si tratta di aritmetica senza segno. In caso di overflow, gli operandi non vengono alterati. La trappola di divisione per zero sarà discussa nel par. 11.2.

Se il quoziente intero può essere contenuto in 32 bit, allora si può usare l'istruzione DIVU.L per eseguire la divisione intera con operandi di 32 bit. In effetti, tale istruzione calcola in 32 bit la porzione intera della divisione. Una condizione di overflow è indicata da V = {1}. Quando il divisore è zero, viene generata una trappola di divisione per zero.

Divisione con segno. L'istruzione DIVS.W opera allo stesso modo dell'istruzione DIVU.W, ma gli operandi sono interi con segno. La convenzione della Motorola è che il segno dell'eventuale resto è uguale al segno del dividendo. Quindi l'istruzione

```
DIVS.W      #3,D1
```

con (D1) = \$FFFF FFF6, calcola – 10/3, producendo il risultato:

(D1) = \$FFFF FFFD

cioè Q = –3 e R = –1. Il codice di condizione N è posto a {1} per indicare che il quoziente è negativo.

Nella divisione con segno, il quoziente può estendersi da -2^{15} a $+2^{15} + 1$. Pertanto, si verificherà un overflow, a meno che la grandezza del dividendo di 32 bit non sia minore di 2^{15} volte rispetto a quella del divisore. Il bit V viene posto a {1} se si presenta un overflow. Se il divisore è zero, viene generata una trappola.

L'istruzione DIVS.L calcola il quoziente di 32 bit ottenuto dalla divisione di due operandi con segno di 32 bit. Qualsiasi resto viene scartato e il codice di condizione V viene posto a {1} se si presenta un overflow. Con questa istruzione, come pure con l'istruzione DIVU.L, deve verificarsi la condizione matematica:

$$0 < |\text{divisore}| < |\text{dividendo}|$$

Se il risultato è tale che:

$$|divisore| * |quoziente| = |dividendo|$$

allora non c'è resto. Qui le sbarrette verticali che racchiudono l'operando indicano il valore assoluto del numero.

Esempio 7-3

Sebbene l'impiego di una singola istruzione MULS.W non possa produrre un overflow, l'uso di queste istruzioni in un'equazione che richiede varie moltiplicazioni potrebbe produrre un risultato che supera la massima grandezza ammissibile. Per esempio, nel calcolo della seguente somma di quadrati:

$$Z = \sum_{i=i}^{n} (X_i^2 + Y_i^2)$$

i singoli prodotti non possono generare un overflow in 32 bit, ma ciò potrebbe avvenire per la somma di più termini. Il programma in Fig. 7.5 calcola la somma dei quadrati di N coppie di interi con segno di 16 bit. I numeri sono memorizzati nell'ordine indicato a pagina seguente.

```
                            1.         TTL      FIGURA 7.5
                            2.         LLEN     100
00010000                    3.         ORG      $10000
                            4.  *
                            5.  *   SOMMA DI QUADRATI
                            6.  *
                            7.  *      INPUT:  (D1.W) = LUNGHEZZA DELLA COLONNA
                            8.  *              (A1.L) = INDIRIZZO DELLA COLONNA DI NUMERI
                            9.  *                       MEMORIZZATA X1,Y1,X2,Y2,...,XN,YN
                           10.  *
                           11.  *      OUTPUT: (D3.L) = RISULTATO
                           12.  *              (D4.W) = 0 : SUCCESSO
                           13.  *                     - 1 : ERRORE
                           14.  *
00010000  48E7 6040        15.  SUMSQ  MOVEM.L D1/D2/A1,-(SP)  ;SALVA I REGISTRI
00010004  4283             16.         CLR.L   D3              ;SOMMA := 0
00010006  383C FFFF        17.         MOVE.W  #-1,D4          ;FISSA STATO NORMALE A ERRORE
0001000A  5341             18.         SUBQ.W  #1,D1           ;SE LA LUNGHEZZA E' ZERO
0001000C  6D00 0020        19.         BLT     FINE            ;ALLORA ESCE CON ERRORE
                           20.  *
00010010  3411             21.  LOOP   MOVE.W  (0,A1),D2       ;CALCOLA
00010012  C5C2             22.         MULS.W  D2,D3           ;XN**2
00010014  D682             23.         ADD.L   D2,D3           ;AGGIUNGE ALLA SOMMA
00010016  6900 0016        24.         BVS     FINE            ;SE ERRORE, ESCE
0001001A  3429 0002        25.         MOVE.W  (2,A1),D2       ;CALCOLA
0001001E  C5C2             26.         MULS.W  D2,D2           ;YN**2
00010020  D682             27.         ADD.L   D2,D3           ;AGGIUNGE ALLA SOMMA
00010022  6900 000A        28.         BVS     FINE            ;SE ERRORE, ESCE
00010026  5889             29.         ADD.L   #4,A1           ;INCREMENTA ALLA COPPIA SUCCESSIVA
00010028  51C9 FFE6        30.         DBRA    D1,LOOP         ;DECREMENTA IL CONTATORE
                           31.  *                              ;  E CONTINUA FINCHE' -1
                           32.  *
0001002C  4244             33.         CLR.W   D4              ;IMPOSTA STATO PER SUCCESSO
0001002E  4CDF 0206        34.  FINE   MOVEM.L (SP)+,D1/D2/A1  ;RIPRISTINA I REGISTRI
00010032  4E75             35.         RTS
00010034                   36.         END
```

Fig. 7.5 *Programma per la somma di quadrati.*

X(1), Y(1), X(2), Y(2), ..., X(N), Y(N)

Si tratta di un vettore o colonna di word di 16 bit il cui primo indirizzo è fornito da (A1) allorché il programma viene inserito. Si presume che la lunghezza N sia specificata nei 16 bit meno significativi di D1. Il risultato di 32 bit viene accumulato in D3 finché non si presenta un errore. Dopo l'esecuzione, un errore è indicato da (D4)[15:0] = –1 e qualsiasi programma che utilizza il risultato dovrebbe verificare lo stato di D4 prima di accettare il risultato. Per indirizzare gli operandi s'impiega il modo d'indirizzamento indiretto di registro d'indirizzo con spostamento, cosicché, se si presenta un overflow, (A1) punta alla coppia corrente di operandi, come si potrebbe rilevare dalla "traccia" del programma per il debugging. Si potrebbe usare l'indirizzamento di postincremento con A1 per eliminare l'istruzione ADD che incrementa A1 di 4 per puntare alla successiva coppia di operandi.

Il resto nella divisione. Si consideri la divisione senza segno:

$$Y/W = Q + R/W$$

dove R/W è il resto, che dev'essere minore di 1. Pertanto, R/W ha la rappresentazione:

$$d_{-1} \times 10^{-1} + d_{-2} \times 10^{-2} + \cdots$$

ed il numero in notazione posizionale può essere scritto come:

$$Q \cdot d_{-1} d_{-2} \cdots$$

fintantoché l'operazione di divisione non causa un overflow. Se si considera solo la parte frazionaria, la moltiplicazione di R/W per 10 fornisce d_{-1} come primo intero, con un resto di

$$d_{-2} \times 10_{-1} + \cdots$$

Le successive moltiplicazioni di R per 10, seguite da una divisione per W, forniscono le cifre decimali del quoziente per il numero desiderato di posizioni. Ad esempio, 22/7 vale 3.142..., che approssima π fino a tre cifre decimali. Le divisioni e le moltiplicazioni forniscono:

$$22/7 = 3 + 1/7$$
$$10/7 = 1 + 3/7$$
$$30/7 = 4 + 2/7$$
$$20/7 = 2 + 6/7$$

e così via finché non è stato calcolato il risultato 3.142... Naturalmente, le operazioni saranno svolte in binario nel computer, ma ogni cifra può essere convertita in decimale codificato in binario (BCD) o in ASCII, a seconda di ciò che si desidera per l'uscita. Gli esempi nel par. 7.6 riguarderanno tali conversioni.

Esempio 7-4

La subroutine mostrata nella Fig. 7.6 calcola la media di una serie di numeri memorizzati in forma di vettore o colonna e indirizzati da A1. Se il numero di valori, che è contenuto in D1, non è zero, allora la somma di 32 bit sarà formata in D3. Tale somma sarà poi divisa per il numero di valori e la word meno significativa di D3 conterrà il quoziente, mentre l'eventuale resto sarà contenuto nella word più significativa. In questo programma non c'è alcun test per l'overflow.

```
                       1.          TTL     FIGURA 7.6
                       2.          LLEN    100
00010000               3.          ORG     $10000
                       4.  *
                       5.  *   CALCOLO DI MEDIA
                       6.  *
                       7.  *       INPUT:  (A1.L) = INDIRIZZO DI COLONNA DI NUMERI DI 16 BIT
                       8.  *               (D1.W) = LUNGHEZZA DELLA COLONNA
                       9.  *
                      10.  *       OUTPUT: (D3)[15:0]  = MEDIA
                      11.  *               (D3)[31:16] = RESTO DI SOMMA/LUNGHEZZA
                      12.  *
00010000 48E7 6840    13.  AVG     MOVEM.L D1-D2/D4/A1,-(SP)  ;SALVA I REGISTRI
                      14.  *
00010004 4A41         15.          TST.W   D1              ;SE LUNGHEZZA = 0
00010006 6700 0014    16.          BEQ     FINE            ;  ALLORA TERMINA
                      17.  *
0001000A 4282         18.          CLR.L   D2
0001000C 4283         19.          CLR.L   D3              ;SOMMA = 0
0001000E 3801         20.          MOVE.W  D1,D4           ;PREDISPONE CONTATORE
00010010 5344         21.          SUBQ    #1,D4           ;  A LUNGHEZZA - 1
                      22.  *
00010012 3419         23.  LOOP    MOVE.W  (A1)+,D2        ;CICLO PER LA SOMMA DI NUMERI
00010014 D682         24.          ADD.L   D2,D3
00010016 51CC FFFA    25.          DBRA    D4,LOOP
                      26.  *
0001001A 87C1         27.          DIVS.W  D1,D3           ;SOMMA/LUNGHEZZA
                      28.  *
0001001C 4CDF 0216    29.  FINE    MOVEM.L (SP)+,D1-D2/D4/A1  ;RIPRISTINA I REGISTRI
00010020 4E75         30.          RTS
00010022              31.          END
```

Fig. 7.6 *Programma per la media di valori.*

7.3.1 ESERCIZI

Si determini il quoziente e il resto nelle seguenti divisioni quando le istruzioni elencate sono eseguite col dividendo e il divisore come mostrato.

(a) 10/5; DIVU.W
(b) −10/5; DIVU.W
(c) −10/5; DIVS.W
(d) −5/2; DIVS.W

I valori negativi dovrebbero essere scritti nella notazione in complemento a 2 per effettuare le divisioni.

7.3.2 Si supponga che due interi con segno siano moltiplicati dall'istruzione MULU. Si dimostri che la moltiplicazione binaria senza segno causerà un errore se uno o entrambi i numeri sono negativi. Si provi moltiplicando (−1) x (−1) nella notazione in complemento a 2 ma con la moltiplicazione senza segno. Come si può correggere il risultato?

7.3.3 Se $N_2 < 2^m \times N_1$ nella divisione binaria senza segno N_2/N_1, si dimostri che non si può verificare un overflow se il dividendo ha $2m$ bit ed il quoziente è di m bit.

7.3.4 Si scriva una routine per calcolare un quoziente di 32 bit ed un resto di 32 bit quando avviene un overflow con l'istruzione DIVU.W. Il risultato può essere ottenuto scrivendo Y/W come:

$(Y2 \times 2^{16} + Y1)/W$

dove Y2 denota i 16 bit superiori del dividendo, mentre Y1 rappresenta i 16 bit inferiori. Si confronti il risultato con quello dell'istruzione DIVU.L, che sarà presentata nel prossimo paragrafo.

7.3.5 Si determini il risultato in (D1) ed i valori dei codici di condizione per ciascuna istruzione elencata di seguito, quando (D1) = $FFFFFFFE prima dell'esecuzione.

(a) MULU.L #2,D1
(b) MULS.L #2,D1

7.3.6 Si determini il quoziente e l'eventuale resto per le seguenti istruzioni, col dividendo e il divisore mostrati.

(a) 3/5; DIVU.L
(b) 21/5; DIVU.L
(c) −3/2; DIVS.L
(d) 10/-3; DIVS.W
(e) 3/−2; DIVS.L

7.4 ARITMETICA DI INTERI IN PRECISIONE MULTIPLA

Nelle misurazioni scientifiche, il termine *accuratezza* si riferisce alla correttezza di una misura, cioè al fatto di essere esente da errori. Invece, la *precisione* denota la quantità di dettaglio impiegata per rappresentare una misura. Per valori numerici, la quantità di precisione è di solito espressa specificando il numero di

cifre significative nel valore numerico. Se una quantità è giudicata di precisione insufficiente per una certa applicazione, si possono usare cifre significative addizionali per produrre un risultato più preciso.

Le unità aritmetiche nei microprocessori operano su un massimo di *m* cifre quando vengono eseguite operazioni aritmetiche. Questa lunghezza massima sarà denominata lunghezza di *singola precisione*. La massima lunghezza di singola precisione nell'MC68020 è di 32 bit, ma possono essere gestite anche quantità di 8 bit e di 16 bit. Sequenze di lunghezza maggiore non possono essere trattate come un singolo operando aritmetico dal processore. Pertanto, al fine di estendere la precisione, più operandi di *m* cifre possono essere considerati matematicamente come un singolo valore. Se fossero combinati *k* operandi, il valore sarebbe lungo *k* x *m* cifre. Per esempio, i valori in doppia precisione hanno *k* = 2. Quindi la lunghezza di doppia precisione dell'MC68020 sarebbe 2 x 32, cioè 64 bit.

Le operazioni aritmetiche con operandi in precisione multipla sono eseguite usando le istruzioni del processore su ciascuna porzione di *m* cifre dei valori e poi combinando i risultati. Questo procedimento fornisce la risposta corretta purché siano trattati correttamente quei dettagli matematici, quali i riporti, tra i risultati intermedi.

L'MC68020 dispone di istruzioni speciali per facilitare l'addizione, la sottrazione, la negazione, la divisione e la moltiplicazione di interi in doppia precisione. Il primo dei seguenti sottoparagrafi riguarda soprattutto l'addizione e la sottrazione di due valori di 32 bit, per fornire risultati in doppia precisione di 64 bit. I sottoparagrafi successivi tratteranno la moltiplicazione e la divisione.

7.4.1 Addizione e sottrazione

Le istruzioni ADDX (*ADD with eXtend:* somma con estensione), SUBX (*SUBtract with eXtend:* sottrai con estensione), NEGX (*NEGate with eXtend:* nega con estensione), sono definite nella Tab. 7.4. La differenza tra queste istruzioni estese e le istruzioni per l'addizione, la sottrazione e la negazione discusse in precedenza sta nell'impiego dei bit dei codici di condizione X e Z da parte delle operazioni estese.

Come mostrato nella Tab. 7.5, le istruzioni estese utilizzano il bit X (*eXtend:* estensione) nelle loro operazioni. Se il bit X fosse stato attivato da una precedente operazione, allora le istruzioni ADDX, SUBX e NEGX ne terrebbero conto durante la loro esecuzione. L'impiego primario del bit di estensione è quello di aggiungere un riporto (ADDX) o sottrarre un riporto negativo (SUBX) quando devono essere trattati gli *m* bit superiori di valori in doppia precisione. Il riporto (positivo o negativo) dovrebbe essere stato generato dall'operazione in singola precisione sugli *m* bit inferiori. Per esempio, la sequenza

```
ADD.L     D1,D3
ADDX.L    D2,D4
```

Tab. 7.4 *Istruzioni aritmetiche estese.*

Sintassi	Modalità d'indirizzamento	
	Sorgente	**Destinazione**
Somma o sottrazione estesa ADDX.<l> <Dm>,<Dn> SUBX.<l> <Dm>,<Dn> ADDX.<l> –(Am),–(An) SUBX.<l> –(Am),–(An)	<Dm> Predecremento	<Dn> Predecremento
Negazione con estensione NEGX.<l> <EA>	---	Alterabile di dati

Nota: <l> denota B, W o L.

somma il valore in doppia precisione contenuto in D2/D1 al valore di 64 bit contenuto in D4/D3. Al bit X viene assegnato il medesimo valore del bit C. In generale, la maggior parte delle istruzioni dell'MC68020 che non sono usate per operazioni aritmetiche non agiscono sul bit X, per cui i bit X e C non dovrebbero essere considerati identici. Per esempio, se D4 nell'esempio appena illustrato fosse stato sottoposto ad un test di zero dall'istruzione:

```
TST.L    D4
```

allora il bit di riporto sarebbe azzerato, ma il bit X (impostato dall'istruzione ADDX) resterebbe immutato.

Tab. 7.5 *Operazioni di istruzioni estese.*

ADDX.<l>	<sorg>,<dest>	(dest)[l] ← (sorg)[l] + (dest)[l] + X
SUBX.<l>	<sorg>,<dest>	(dest)[l] ← (dest)[l] – (sorg)[l] – X
NEGX.<l>	<EA>	(EA)[l] ← 0 – (EA)[l] – X

Note:
1. I bit dei codici di condizione C, N e V sono impostati come per qualsiasi operazione aritmetica.
2. Z viene azzerato se il risultato è diverso da zero; altrimenti, resta invariato.
3. X viene uguagliato al bit C.
4. <l> denota B, W o L.
5. [l] indica i bit corrispondenti nell'operazione.

Anche il bit del codice di condizione zero, Z, è trattato in modo speciale dalle istruzioni estese. L'impostazione di Z, dopo che un'istruzione estesa è stata eseguita, dipende sia dalla precedente impostazione di Z che dal valore dell'operando corrente. Si consideri l'intero di 64 bit:

$$0000\ 0000\ 0000\ 0001_{16}$$

in cui il valore dei 32 bit inferiori è diverso da zero. Se ciascuna metà di 32 bit viene sottoposta separamente ad un test di zero, allora Z sarebbe posto a {0} per la porzione meno significativa. Tuttavia, Z diverrebbe {1} dopo l'esame della porzione più significativa. Se successivamente venisse effettuato un test di condizione, il risultato sarebbe basato su un valore zero!

Al fine di ottenere i risultati corretti per le condizioni di doppia precisione, le istruzioni ADDX, SUBX e NEGX impostano il bit Z secondo l'equazione logica:

$$Z = Z_2 \text{ AND } Z_1$$

in cui Z_1 e Z_2 devono valere *entrambi* {1} affinché Z = {1}. Qui Z_1 è il valore di Z prima dell'esecuzione dell'istruzione estesa, mentre Z_2 è il risultato che deriva dall'operazione estesa. Si presume che ciò riguardi la porzione più significativa di un operando in doppia precisione, come nella sequenza di istruzioni appena vista per l'addizione. Quindi, se Z_1 = {0}, allora Z = {0} indipendentemente dal valore assunto da Z_2. Pertanto, Z sarà posto a {1} soltanto se entrambe le porzioni del valore in doppia precisione sono nulle. Nell'esempio del numero di 64 bit, Z = {0} indica che il risultato è diverso da zero quando il valore viene calcolato usando le istruzioni estese.

Un intero in doppia precisione può essere scritto in notazione posizionale come segue:

$$(b_{2m-1}b_{2m-2}\cdots b_m b_{m-1}\cdots b_0)$$

dove le cifre $b_{m-1}b_{m-2}\cdots b_1 b_0$ rappresentano la lunghezza di singola precisione.

Due operandi in doppia precisione N_1 e N_2 possono essere scritti come:

$$N_1 = N_{1U} + N_{1L}$$
$$N_2 = N_{2U} + N_{2L}$$

dove N_{iL} indica le cifre da 0 a $m - 1$, mentre N_{iU} denota le cifre da m a $2m - 1$, con $i = 1$ o 2. Questa notazione sarà impiegata, laddove necessario, per distinguere le porzioni di precisione inferiore da quelle di precisione superiore di un valore.

Le sequenze di istruzioni per eseguire l'addizione o la sottrazione in doppia precisione richiedono che l'operazione ADD sia seguita da ADDX e che SUB sia seguita da SUBX. Per l'addizione, un eventuale riporto generato dalla somma (ADD) della porzione inferiore è indicato da entrambi i bit C e X dei codici di condizione. La somma superiore viene quindi calcolata da ADDX, che aggiunge il bit X al risultato. Un riporto generato allorché viene eseguita l'istruzione ADDX indica un risultato senza segno che è troppo grande per poter essere rappresentato in dop-

pia precisione. Se devono essere rappresentati interi con segno, una condizione di overflow sarebbe indicata dal bit di codice di condizione V = {1}.

Quando vengono sottratti interi in doppia precisione, un eventuale riporto negativo generato dalla sottrazione delle porzioni meno significative è indicato dal bit X. Questo sarà sottratto dalla differenza dei valori più significativi. Una condizione di fuori-intervallo per la porzione più significativa è indicata, dopo l'esecuzione di SUBX, da C = {1} per interi senza segno, oppure da V = {1} se fossero sottratti interi con segno.

La sequenza di istruzioni

NEG
NEGX

effettua la negazione di un intero in doppia precisione quando NEG opera sulla porzione di precisione inferiore mentre NEGX opera sulla porzione di precisione superiore. Una condizione di overflow si presenta se viene negato l'intero più negativo.

Esempio 7-5

Alcuni esempi di operazioni in precisione multipla sono illustrati nella Fig. 7.7. Per semplicità, le lunghezze di singola precisione sono di otto cifre binarie. Sono mostrati anche gli stati dei codici di condizione pertinenti dopo ciascuna porzione dell'operazione in precisione multipla. In ogni caso, sono trattati per primi gli 8 bit inferiori degli operandi; dopodiché, viene eseguita l'istruzione estesa che opera sugli 8 bit superiori.

Istruzione	Addizione binaria
ADD.B	1000 0000 + 1000 0000 1) 0000 0000 ; X = {1}, Z = {1}
ADDX	0100 0000 + 0001 0000 + 1 (0101 0001 0000 0000)$_2$; X = {0}, Z = {0}

Istruzione	Sottrazione binaria
SUB.B	0000 0000 − 1111 1111 1) 0000 0001 ; X = {1}, Z = {0}
SUBX	0000 0011 − 0000 0000 − 1 (0000 0010 0000 0001)$_2$; X = {0}, Z = {0}

Fig. 7.7 *Operazioni in precisione multipla.*

Esempio 7-6

La subroutine mostrata nella Fig. 7.8 somma gli elementi di due colonne o vettori di *N* interi senza segno, elemento per elemento. Se X[i] rappresenta le locazioni dell'*i*-esimo elemento del primo vettore, mentre Y[i] è l'elemento corrispondente del secondo, l'operazione è:

$$(Y[i]) \leftarrow (X[i]) + (Y[i])$$

con $i = 1, 2, \ldots, N$. Nella memoria, ogni intero di 64 bit è memorizzato con i 32 bit meno significativi all'indirizzo *superiore* di due locazioni di longword. I valori sono memorizzati con l'ultimo elemento, X[N] o Y[N], all'indirizzo di memoria più basso ed ogni array richiede 2*N* locazioni di longword, cioè 8*N* byte. Questo metodo di memorizzazione trae vantaggio dalla capacità d'indirizzamento con predecremento dell'MC68020 che impiega le istruzioni estese.

All'entrata nella subroutine, A1 e A2 dovrebbero puntare alla successiva locazione di longword che segue il primo ed il secondo vettore, rispettivamente. Il registro d'indirizzo A3 deve contenere l'indirizzo dell'ultimo elemento nel secondo vettore (cioè, esso dovrebbe puntare a Y[N]).

```
                         1.          TTL     FIGURA 7.8
                         2.          LLEN    100
00010000                 3.          ORG     $10000
                         4.  *
                         5.  *   SOMMA DUE VETTORI DI INTERI SENZA SEGNO DI 64 BIT
                         6.  *       Y(I) <-- Y(I) + X(I)       FOR I = 1,N
                         7.  *
                         8.  *       INPUT:  (A1.L) = ULTIMO INDIRIZZO DI PRIMO VETTORE + 4
                         9.  *               (A2.L) = ULTIMO INDIRIZZO DI SECONDO VETTORE + 4
                        10.  *               (A3.L) = INDIRIZZO DELL'ULTIMO ELEMENTO
                        11.  *                        NEL SECONDO VETTORE (INDIRIZZO DI Y(N))
                        12.  *
                        13.  *       OUTPUT: (A2.L) = INDIRIZZO DI SOMME
                        14.  *               (D1.B) = 0 : RIVELATO ERRORE
                        15.  *                        NON 0 : SUCCESSO
                        16.  *
                        17.  *  NOTE:
                        18.  *    1. I NUMERI DI 64 BIT SONO MEMORIZZATI COME SEGUE:
                        19.  *
                        20.  *       XN [63:32]         MEMORIA BASSA
                        21.  *       XN [31:0]     INDIRIZZO DI PRIMA LONGWORD
                        22.  *       .
                        23.  *       .
                        24.  *       .
                        25.  *       X1 [63:32]         MEMORIA ALTA
                        26.  *       X1 [31:0]     INDIRIZZO DI ULTIMA LONGWORD
                        27.  *
                        28.  *    2. A1 E A2 PUNTANO A X1+4
                        29.  *       A3 PUNTA A XN [63:32]
                        30.  *
00010000  48E7 8060     31.  SERIES  MOVEM.L D0/A1-A2,-(SP)  ;SALVA I REGISTRI
00010004  4201          32.          CLR.B   D1              ;FISSA STATO PER FALLIMENTO
00010006  2021          33.  LOOP    MOVE.L  -(A1),D0
00010008  D1A2          34.          ADD.L   D0,-(A2)
0001000A  D589          35.          ADDX.L  -(A1),-(A2)
0001000C  6500 000A     36.          BCS     ERROR           ;OVERFLOW
00010010  B7CA          37.          CMPA.L  A2,A3           ;SE NON ULTIMO NUMERO
00010012  65 F2         38.          BCS     LOOP            ;ALLORA CONTINUA
                        39.  *                               ;  ALTRIMENTI TERMINA
00010014  123C 00FF     40.          MOVE.B  #-1,D1          ;FISSA STATO PER SUCCESSO
00010018  4CDF 0601     41.  ERROR   MOVEM.L (SP)+,D0/A1-A2  ;RIPRISTINA I REGISTRI
0001001C  4E75          42.          RTS
0001001E                43.          END
```

Fig. 7.8 *Programma di addizione in precisione multipla.*

Se non si verifica alcun overflow, le addizioni continueranno finché non sarà verificata la condizione (A2) = (A3) per indicare la locazione dell'ultimo valore da sommare. L'istruzione di confronto di indirizzo (*CoMPare Address:* CMPA) potrebbe modificare il codice di condizione C ma lascia X immutato. Quando i due indirizzi sono uguali, il test per il salto è FALSO e il ciclo termina. In programmi in cui lo stato del bit X dev'essere preservato ma il bit C è usato per i test di condizione, il fatto di avere i bit C e X separati è un vantaggio, poiché il bit X non dev'essere salvato prima delle operazioni di confronto.

7.4.2 Moltiplicazione

Le istruzioni MULU.W e MULU.L e le loro controparti con segno MULS.W e MULS.L forniscono prodotti di 32 bit. MULU.L e MULS.L possono causare un overflow se il prodotto dei due operandi di 32 bit supera la lunghezza di 32 bit. In questo caso, è richiesto un prodotto di 64 bit e si dovrebbero usare le istruzioni mostrate nella Tab. 7.6. Per esempio, l'istruzione

```
MULS.L      D0,D6:D7
```

moltiplica (D0)*(D7) e lascia il prodotto di 64 bit in (D6):(D7). In base alla sintassi nella Tab. 7.6, (D6) contiene la porzione più significativa del risultato. L'operando di sorgente, contenuto in D0 in quest'esempio, può essere specificato da qualsiasi modalità d'indirizzamento, tranne quella diretta di registro d'indirizzo. Quindi l'operando di sorgente può essere contenuto in una longword di memoria di 32 bit indirizzata con qualsiasi modalità oppure in un registro di dati. Se l'operando di sorgente è considerato il moltiplicatore, allora il moltiplicando di 32 bit dev'essere contenuto nel registro meno significativo <Dl>, come definito nella Tab. 7.6 L'unica restrizione sull'impiego di registri è che Dh e Dl non dovrebbero essere il medesimo registro per moltiplicazioni di 64 bit. La notazione Dl indica la porzione meno significativa di un valore, mentre Dh indica la porzione più significativa.

Tab. 7.6 *Istruzioni di moltiplicazione di 64 bit.*

Sintassi	Operazione
Con segno MULS.L <EA>,<Dh>:<Dl> *Senza segno* MULU.L <EA>,<Dh>:<Dl>	(Dh)[31:0]:(Dl)[31:0] ← (<EA>*(Dl)[31:0]

Note:
1. <Dh> e <Dl> sono qualsiasi coppia di registri di dati.
2. <EA> è specificato da qualsiasi modo d'indirizzamento di dati.
3. <EA> contiene un operando di 32 bit mentre (Dl)[31:0] contiene l'altro. I 32 bit meno significativi del prodotto sono contenuti in (Dl) mentre i 32 bit più significativi sono contenuti in (Dh).

Esempio 7-7

La Fig. 7.9 illustra la moltiplicazione di due interi di 32 bit in esadecimale, che si può impiegare come un risultato per il test di routine di moltiplicazione di 64 bit. Le somme parziali sono mostrate allineate per favorire la leggibilità.

```
3333 4444
1111 2222
= 369D4C3A8641908

= 3333 × 1111 = 369C963
= 4444 × 1111 = 48D0C84
= 3333 × 2222 = 6D392C6
= 4444 × 2222 = 91A1908

369 | D4C3 | A864 | 1908
```

Fig. 7.9 *Esempio di moltiplicazione in doppia precisione.*

Esempio 7-8

La subroutine della Fig. 7.10 moltiplica due vettori di numeri interi di 32 bit, elemento per elemento, per formare un vettore di prodotti di 64 bit. Nell'equazione

Z(i) = X(i) * Y(i)

X(i) e Y(i) sono vettori di N valori di 32 bit. I prodotti di 64 bit sono contenuti nella memoria, nell'array definito da Z(i). Gli indirizzi iniziali di X, Y e Z sono tenuti in A1, A2 e A3, rispettivamente. Il registro di dati D3 contiene il numero di interi N in ciascun vettore. La locazione di word quadrupla (*quad word*) Z(i) contiene Z(i)[63:32], Z(i)[31:0], con i = 1, 2, ..., N.

```
                            1.          TTL     FIGURA 7.10
                            2.          LLEN    100
00010000                    3.          ORG     $10000
                            4.   *
                            5.   *   MOLTIPLICAZIONE DI VETTORI DI INTERI
                            6.   *   SENZA SEGNO, DI 32 X 32 BIT
                            7.   *   ELEMENTO PER ELEMENTO
                            8.   *
                            9.   *      Z(I) = X(I) * Y(I) ; PRODOTTO DI 64 BIT
                           10.   *
                           11.   *      INPUT:  (D3.L) = NUMERO DI VALORI
                           12.   *              (A1.L) = INDIRIZZO DEL VETTORE X
                           13.   *              (A2.L) = INDIRIZZO DEL VETTORE Y
                           14.   *              (A3.L) = INDIRIZZO DEL PRODOTTO Z
                           15.   *
                           16.   *      OUTPUT: Z(I)
                           17.   *              MEMORIZZATO COME
                           18.   *              Z(I)[63:32]
                           19.   *              Z(I)[31:0]
                           20.   *
                           21.   ************
00010000  48E7 F000        22.          MOVEM.L D0-D3,-(SP)     ;SALVA I REGISTRI
00010004  5343             23.          SUBQ    #1,D3           ;N-1 IN D3
                           24.   *
00010006  2019             25.   LOOP   MOVE.L  (A1)+,D0        ;X
00010008  221A             26.          MOVE.L  (A2)+,D1        ;Y
0001000A  4C00 1402        27.          MULU.L  D0,D2:D1        ;Z=X*Y
                           28.   *
0001000E  26C2             29.          MOVE.L  D2,(A3)+        ;Z ALTO
00010010  26C1             30.          MOVE.L  D1,(A3)+        ;Z BASSO
00010012  51CB FFF2        31.          DBRA    D3,LOOP         ;MOLTIPLICA N NUMERI
                           32.   *
00010016  4CDF 000F        33.          MOVEM.L (SP)+,D0-D3     ;RIPRISTINA I REGISTRI
0001001A  4E75             34.          RTS                     ; E RITORNA
```

Fig. 7.10 *Routine di moltiplicazione in doppia precisione.*

Esempio 7-9

Le istruzioni dell'MC68020 MULU e MULS formano un prodotto di 32 bit o di 64 bit quando due numeri vengono moltiplicati. Al fine di moltiplicare operandi di 64 bit per ottenere un prodotto di 128 bit, si può usare ripetutamente l'istruzione di moltiplicazione per formare i prodotti parziali. Questi prodotti parziali saranno sommati per produrre il risultato. Per esempio, si consideri la moltiplicazione

$$(x + y) \times (w + z) = x \times w + y \times w + x \times z + y \times z$$

che richiede quattro moltiplicazioni e quattro addizioni. La moltiplicazione in doppia precisione è teoricamente simile se y e z rappresentano i valori di precisione inferiore degli operandi, mentre x e w indicano i valori di precisione superiore. Tuttavia, nei calcoli svolti dalla macchina, si deve tener conto del fatto che gli intervalli di grandezza dei diversi prodotti parziali non sono identici. Il calcolo appropriato può essere determinato per numeri senza segno scrivendo l'operando in doppia precisione nella forma:

$$N = N_U \times 2^m + N_L$$

Questo è un numero di $2m$ cifre, in cui N_U e N_L sono gli interi di m cifre formati dalle porzioni superiore e inferiore, rispettivamente. Il prodotto di due interi in doppia precisione N_1 e N_2 diviene allora:

$$N_1 \times N_2 = 2^{2m} \times (N_{2U} \times N_{1U}) + 2^m \times (N_{2L} \times N_{1U} + N_{2U} \times N_{1L}) + N_{2L}N_{1L}$$

La lunghezza totale del prodotto è di $4m$ cifre ed ogni prodotto parziale ha $2m$ cifre. L'algortimo di macchina che esegue l'operazione di moltiplicazione deve allineare correttamente i prodotti parziali prima di sommarli, con un procedimento simile a quello seguito per la moltiplicazione manuale. Qualsiasi riporto da un risultato meno significativo ad uno più significativo dev'essere aggiunto correttamente.

Quando vengono moltiplicati interi con segno in precisione multipla, il metodo appena descritto per gli interi senza segno non è applicabile se uno o entrambi gli operandi da moltiplicare sono negativi. Un'investigazione matematica usando la rappresentazione in complemento a 2 descritta nel cap. 3 fornisce un algoritmo che è adatto a questo caso. Il metodo è discusso in vari riferimenti bibliografici relativi a questo capitolo, elencati nell'app. E alla fine del libro. Un metodo alternativo consiste nel modificare il segno di qualsiasi operando negativo (cioè, negarlo) e poi correggere il segno dopo aver eseguito la moltiplicazione senza segno. Il programma che svolge tale funzione è lasciato come esercizio per il lettore.

7.4.3 Divisione

Le istruzioni di divisione con e senza segno dell'MC68020 includono due forme che permettono il calcolo di un quoziente di 32 bit con un resto di 32 bit. Come mostrato nella Tab. 7.7, una forma (DIVU.L, DIVS.L) consente di dividere un dividendo di 64 bit per un divisore di 32 bit, per fornire un risultato di 64 bit. Le altre istruzioni (DIVSL.L, DIVUL.L) dividono operandi di 32 bit per produrre un risultato di 64 bit. Come per qualsiasi istruzione di divisione, una divisione per zero genera una trappola. Inoltre, se il quoziente non può essere contenuto in 32 bit, viene segnalata una condizione di overflow (V = {1}).

Tab. 7.7 *Istruzioni di divisione in precisione multipla.*

Sintassi	Operazione
DIVS.L <EA>,<Dr>:<Dq> DIVU.L <EA>,<Dr>:<Dq>	(Dq)[31:0] : (Dr)[31:0] = (Dr)[31:0] : (Dq)[31:0] ———————— (<EA>)[31:0]
DIVUL.L <EA>,<Dr>:<Dq> DIVSL.L <EA>,<Dr>:<Dq>	(Dq)[31:0] : (Dr)[31:0] = (Dq)[31:0] ———————— (<EA>)[31:0]

Note:
1. <Dq> contiene i 32 bit meno significativi del dividendo prima della divisione. Il quoziente di 32 bit è posto in questo registro dopo la divisione.
2. <Dr> contiene i 32 bit più significativi del dividendo di 64 bit ed il resto di 32 bit dopo la divisione.
3. Si verifica un overflow se il quoziente è maggiore di un intero di 32 bit. La divisione per zero causa una trappola.
4. <EA> può essere specificato da qualsiasi modalità d'indirizzamento, tranne quella diretta di registro d'indirizzo.

ESERCIZI

7.4.1 Si determini l'intervallo di interi senza segno, di frazioni e di interi con segno per una rappresentazione di 64 bit. Si esprimano le risposte come potenze di 10.

7.4.2 Si tracci un diagramma dell'operazione necessaria per eseguire una moltiplicazione di 64 bit x 64 bit = 128 bit usando l'istruzione MULU. Si scriva la subroutine e la si provi.

7.4.3 Si determini il risultato di

```
MULU.L     #2,D0:D1
```

se (D1) = $FFFFFFFE inizialmente.

7.4.4

Sia (D0) = \$FFFFFFFF e (D1) = \$FFFFFFFE. Che risultato si ottiene se le seguenti istruzioni sono eseguite con i valori assegnati di (D0) e (D1)?

```
(a) DIVU.L    #2,D0:D1
(b) DIVS.L    #2,D0:D1
```

7.4.5

Si modifichi la routine di moltiplicazione di 128 bit del problema 7.4.2 per moltiplicare interi in complemento a 2.

7.5 ARITMETICA DECIMALE

L'MC68020 dispone di istruzioni per le operazioni aritmetiche su valori decimali rappresentati in decimale codificato in binario (*Binary-Coded Decimal:* BCD). Questo codice è stato definito nel cap. 3; in questo paragrafo sono applicati molti dei princìpi matematici presentati allora. Le tre istruzioni per l'aritmetica BCD sono definite nella Tab. 7.8: esse consentono l'addizione, la sottrazione e la negazione di valori BCD. Per ogni istruzione la lunghezza dell'operando è di 8 bit, che rappresentano due cifre BCD. Per esempio, l'istruzione di somma decimale con estensione, ABCD (*Add BCD:* somma BCD):

```
ABCD    D1,D2
```

esegue l'addizione decimale tra operandi lunghi un byte. L'operazione è:

$$(D2)[7:0] \leftarrow (D1)[7:0] + (D2)[7:0] + X$$

Si noti che questa istruzione aggiunge il valore del bit di codice di condizione X nella somma per facilitare le addizioni in precisione multipla. Comunque, il bit X dev'essere azzerato prima che sia eseguita la prima istruzione ABCD. Dopo l'operazione di addizione, X = {1} indica che si è avuto un riporto decimale poiché la somma era maggiore di 99. Il bit Z viene azzerato se la somma non è zero; altrimenti, esso rimane invariato per consentire di effettuare il test di zero dopo le operazioni in precisione multipla. L'istruzione di sottrazione decimale con estensione, SBCD (*Subtract BCD:* sottrai BCD) opera in maniera simile, ma l'operando di sorgente ed il valore del bit X vengono sottratti entrambi dal valore di destinazione.

Le operazioni delle istruzioni BCD sono simili a quelle per l'aritmetica in precisione estesa, per quanto concerne i bit dei codici di condizione. L'eccezione sta nel fatto che i bit N e V non sono definiti per le operazioni BCD. Inoltre, le istruzioni BCD limitano la lunghezza dell'operando a 8 bit, mentre le modalità d'indirizzamento ammesse sono soltanto quella diretta di registro di dati o con predecremento per l'addizione e la sottrazione BCD. L'istruzione di negazione di decimale con estensione, NBCD (*Negate BCD:* nega BCD) forma il complemento a 10 di un operando

Tab. 7.8 *Istruzioni di aritmetica decimale.*

Sintassi	Operazione
Addizione	
ABCD <Dm>,<Dn>	(Dn)[7:0] ← (Dn)[7:0] + (Dm)[7:0] + X
ABCD –(Am),–(An)	(dest) ← (dest) + (sorg) + X
Sottrazione	
SBCD <Dm>,<Dn>	(Dn)[7:0] ← (Dn)[7:0] – (Dm)[7:0] – X
SBCD –(Am),–(An)	(dest) ← (dest) – (sorg) – X
Negazione	
NBCD <EA>	(EA) ← 0 – (EA) – X

Note:
1. Tutte le operazioni eseguono l'aritmetica decimale su due cifre BCD.
2. I bit dei codici di condizione N e V sono indefiniti.
3. C viene posto a {1} se avviene un riporto (positivo o negativo) decimale.
4. Z viene posto a {0} se il risultato è diverso da zero; altrimenti, rimane immutato.

di 2 cifre quando X = {0} prima dell'operazione. Se X = {1}, allora NBCD esegue il complemento a 9. Essa consente qualsiasi modalità d'indirizzamento alterabile di dati per l'indirizzo effettivo nella forma simbolica:

```
NBCD        <EA>
```

Ciò esclude la possibilità che un operando sia contenuto in un registro d'indirizzo o che possa essere indirizzato relativamente al contatore di programma.

Esempio 7-10

La Tab. 7.9 mostra gli effetti dell'addizione e della sottrazione decimale per diversi valore degli operandi e dei codici di condizione. L'addizione dei valori 65 e 17 dà 82 se il bit X è azzerato, oppure 83 se il bit X vale {1}. La somma di 42 e 77 fornisce un risultato di 19 con l'indicazione di un riporto. Il valore corretto 119 richiederebbe un'ulteriore cifra BCD. Nella tabella è mostrata anche l'indicazione scorretta che risulta dall'addizione di 0 con 0, in cui il bit Z non viene posto a {1}, come dovrebbe essere.

Tab. 7.9 *Esempio di operazioni BCD.*

(a) Addizione: ABCD (sorg),(dest)

Prima dell'esecuzione				*Dopo l'esecuzione*		
(sorg)	(dest)	X	Z	(dest)	X	Z
65	17	0	1	82	0	0
65	17	1	1	83	0	0
42	77	0	0	19	1	0
0	0	0	0	00	0	0

(b) Sottrazione: SBCD (sorg),(dest)

Prima dell'esecuzione			*Dopo l'esecuzione*	
(sorg)	(dest)	X	(dest)	X
32	77	0	45	0
65	17	0	52	1
65	17	1	51	1
35	35	1	99	1

Nota: I contenuti delle locazioni di sorgente e di destinazione sono valori decimali.

La sottrazione di due cifre BCD fornisce un risultato corretto per numeri senza segno nell'intervallo da 0 a 99 o numeri con segno tra –10 e +9. La sottrazione dei numeri senza segno 77 – 32 fornisce 45, come previsto. Invece, 17 – 65 lascia il risultato in complemento a dieci di –48 (52) con un indicazione di riporto negativo. Se il bit X vale {1} prima dell'operazione, si ottiene il complemento a nove di –48 (51) quando 65 viene sottratto da 17. Quando vengono sottratti due valori uguali e il bit X è posto a {1}, il risultato è 99, cioè il complemento a nove di 0 con un indicazione di riporto negativo.

Aritmetica decimale in precisione multipla. Le operazioni su numeri BCD con più di due cifre sono eseguite normalmente su operandi contenuti nella memoria anziché in un registro. Il motivo è che le istruzioni ABCD e SBCD possono operare solamente sul byte meno significativo di un registro di dati. Se una stringa di cifre decimali è contenuta in un registro di dati, sarebbero richieste le istruzioni di rotazione (presentate nel cap. 8) per far scorrere nel byte meno significativo le cifre da trattare. Ciò si può evitare eseguendo operazioni da memoria a memoria, mediante l'indirizzamento con predecremento. Per esempio, l'istruzione

```
ABCD    -(A1),-(A2)
```

dapprima decrementa (A1) di 1 e poi (A2) di 2. Dopodiché, saranno sommate le due cifre nelle locazioni di byte indirizzate, più il valore del bit X, e il risultato sarà posto nella locazione di destinazione indirizzata da A2. Per eseguire operazioni su numeri lunghi più di due cifre, si può registrare nella memoria una stringa decimale le cui due cifre meno significative sono situate all'indirizzo di byte più alto. Quindi il numero decimale 123456 alla locazione $1000 sarebbe memorizzato come segue:

(1000) = 12
(1001) = 34
(1002) = 56

Un'addizione o una sottrazione di questo valore dovrebbe iniziare con l'indirizzo di partenza inizializzato a $1003, quando s'impiega una modalità d'indirizzamento con predecremento.

Esempio 7-11

Il programma in Fig. 7.11 somma due interi BCD di sei cifre. Inizialmente, gli indirizzi degli operandi, come quelli appena descritti, sono memorizzati in A1 e A2. La somma sarà contenuta nella locazione indirizzata da A2. Il bit X dev'essere azzerato e il bit Z dev'essere posto a {1} prima che l'addizione inizi. Se il risultato è diverso da zero, il bit Z sarà azzerato dall'addizione. Se gli interi sono limitati a valori positivi, il bit C indicherà una condizione di overflow dopo le addizioni.

```
                        1.          TTL      FIGURA 7.11
                        2.          LLEN     100
00010000                3.          ORG      $10000
                        4.  *
                        5.  *   ADDIZIONE DI BCD
                        6.  *
                        7.  *       INPUT:  (A1.L) = INDIRIZZO DEL BYTE SUCCESSIVO
                        8.  *                        AL PRIMO NUMERO BCD DI 6 CIFRE
                        9.  *               (A2.L) = INDIRIZZO DEL BYTE SUCCESSIVO
                       10.  *                        AL SECONDO NUMERO BCD DI 6 CIFRE
                       11.  *                           (INDIRIZZO DEL NUMERO + 3)
                       12.  *       OUTPUT: (A2.L) = INDIRIZZO DEL BYTE PIU' SIGNIFICATIVO
                       13.  *                        DEL RISULTATO BCD DI 6 CIFRE
                       14.  *
                       15.  *  NOTE:
                       16.  *    1. I NUMERI BCD SONO MEMORIZZATI CON DUE CIFRE/BYTE
                       17.  *       BCD [6:5]
                       18.  *       BCD [4:3]
                       19.  *       BCD [2:1]
                       20.  *
                       21.  *    2. I REGISTRI D'INDIRIZZO PUNTANO A BCD [2:1] + 1
                       22.  *       PER CUI PUO' ESSERE USATO L'INDIRIZZAMENTO
                       23.  *       CON PREDECREMENTO.
                       24.  *
                       25.  *    3. NESSUN TEST PER L'OVERFLOW.
                       26.  *
                       27.
00010000  48E7 0060    28.  ADDBCD  MOVEM.L A1/A2,-(SP)     ;SALVA I REGISTRI
00010004  44FC 0004    29.          MOVE.W  #4,CCR          ;AZZERA IL BIT X
                       30.  *                               ;PONE A 1 IL BIT Z
00010008  C509         31.          ABCD    -(A1),-(A2)     ;SOMMA I NUMERI BCD
0001000A  C509         32.          ABCD    -(A1),-(A2)       .
0001000C  C509         33.          ABCD    -(A1),-(A2)
0001000E  4CDF 0600    34.          MOVEM.L (SP)+,A1/A2     ;RIPRISTINA I REGISTRI
00010012  4E75         35.          RTS                     ;E RITORNA
00010014               36.          END
```

Fig. 7.11 *Routine di addizione BCD di 6 cifre.*

Si noti che le istruzioni ABCD sono state eseguite in sequenza. Se l'istruzione ABCD è impiegata entro un ciclo, allora il valore del bit X dev'essere salvato prima che sia eseguita qualsiasi operazione come SUBQ per la ripetizione del ciclo, poiché queste istruzioni aritmetiche possono modificare il valore del bit X. Dopo aver eseguito un test sulle condizioni appropriate, il programma dovrà ripristinare il bit X, prima che ABCD sia eseguita nuovamente nel ciclo. Un'istruzione CMP non modificherà il bit X, come spiegato nell'esempio 7.6.

ESERCIZI

7.5.1 Si esprimano i seguenti numeri BCD in binario usando quattro cifre. S'illustri la rappresentazione di macchina usando la notazione in complemento a 10.

(a) +37
(b) −37
(c) −1319

7.5.2 Usando i formati dell'MC68020, si eseguano le seguenti operazioni su interi BCD:

(a) 1754 − 1319
(b) 9375 + 3470

Come possono essere interpretati i risultati della parte (b) quando sono ammessi soltanto interi senza segno di 4 cifre?

7.5.3 Si esegua l'addizione di 127 e 299 impiegando l'aritmetica binaria ma con i valori espressi nella notazione BCD. Si "aggiusti" il risultato binario sommando il valore 6 a qualsiasi cifra maggiore di 9 e si aggiunga il riporo alla cifra immediatamente più significativa. (Le istruzioni BCD dell'MC68020 della Motorola eseguono automaticamente questo "aggiustamento" decimale).

7.5.4 Si scriva una subroutine per sommare o sottrarre due interi BCD lunghi fino a 8 cifre ciascuno. Gli interi con segno sono rappresentati nella notazione in complemento a 10. Qual è l'intervallo decimale degli interi validi? Qual è l'intervallo decimale della loro somma o differenza?

7.5.5 Si scriva una subroutine per moltiplicare due numeri BCD positivi di quattro cifre nel formato dell'MC68020. La routine potrebbe eseguire la moltiplicazione mediante l'addizione ripetuta, oppure si può escogitare un algoritmo per eseguire la moltiplicazione decimale.

7.6 INGRESSO/USCITA E CONVERSIONI

Questo paragrafo riguarda soprattutto le tecniche che consentono ad un programmatore di trasferire dati tra la memoria del sistema di computer ed il terminale dell'operatore. Fortunatamente, molti dettagli di questi trasferimenti sono gestiti da routine che fanno parte del sistema operativo o monitor. Questi dettagli sono presentati nel primo sottoparagrafo 7.6.1, ma non saranno discussi completamente fino al cap. 13. Dopo questa introduzione agli aspetti fisici o hardware dei trasferimenti di ingresso/uscita (*Input/Output:* I/O), sarà definito un certo numero di macro-direttive che chiamano le routine nel monitor (tramite il meccanismo della trappola) per completare i trasferimenti. Infine, saranno presentate le varie conversioni tra valori ASCII, binari e BCD.

La Fig. 7.12 illustra la struttura semplificata del sistema di computer, con particolare evidenza all'hardware e al software del trasferimento di I/O. Il sistema è simile a quello presentato nel cap. 2, ma con l'interfaccia al terminale mostrata come chip periferico. Questo chip effettua la conversione seriale-parallela per l'uscita sullo schermo di visualizzazione. Esso viene programmato dalle routine di controllo del chip di I/O per l'ingresso o l'uscita; in questa discussione, tali routine fanno parte del monitor. Saranno discusse le macro di I/O per chiamare queste routine.

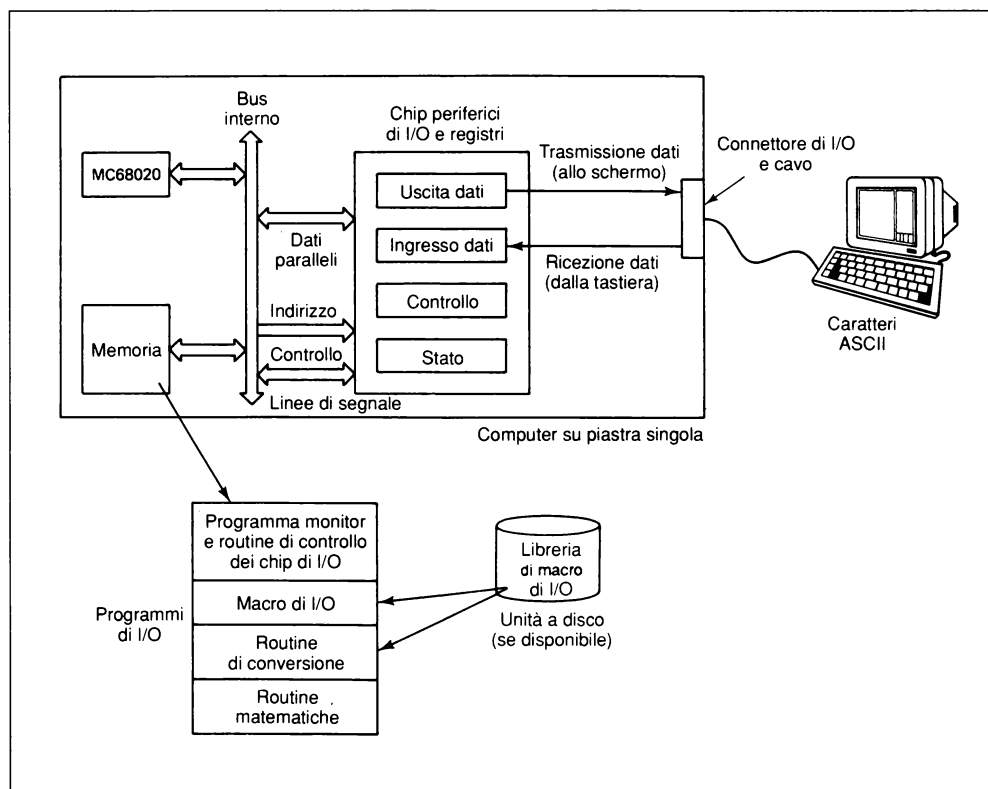



Fig. 7.12 *Diagramma semplificato delle operazioni di ingresso/uscita (input/output: I/O).*

Invece devono essere scritte le routine di conversione, che eseguono appunto la conversione da ASCII a binario o BCD per l'ingresso dei dati. Dopo la conversione, i numeri possono essere trattati come richiesto dal programma applicativo. Viceversa, i risultati di operazioni matematiche contenuti nella memoria devono essere convertiti in caratteri ASCII per la visualizzazione al terminale. Sarebbe comodo creare una libreria di macro di I/O per contenere le routine di conversione e di I/O. Questo modulo sarebbe collegato al programma applicativo quando sono richiesti i trasferimenti di I/O.

Per gli esempi di questo paragrafo è stato impiegato il computer su piastra singola MVME133 con un monitor 133BUG. Altri dettagli concernenti il funzionamento del monitor sono stati presentati nel cap. 5. In quella discussione, il monitor serviva soprattutto per caricare ed eseguire i programmi. La piastra MVME133 è stata presentata nei capp. 1 e 2. Una discussione più completa delle sue capacità e della configurazione dell'hardware sarà presentata nel cap. 13. Per comodità, l'insieme di caratteri ASCII è riportato nell'app. A.

7.6.1 Trasferimento di I/O

Per consentire all'operatore di immettere un carattere dalla tastiera, il chip di I/O nella Fig. 7.12 dev'essere già stato programmato per accettare il carattere. Il procedimento semplificato per la routine di controllo del chip di I/O è la seguente:

(a) Leggere lo stato del chip di I/O dal suo registro di stato.

(b) Se il chip non è occupato, programmare il suo registro di controllo per accettare un carattere. Se occupato, attendere e riprovare.

(c) Una volta che il carattere è stato ricevuto, trasferirlo dal registro d'ingresso dati alla memoria.

Se il chip di I/O è pronto per l'acquisizione dei dati d'ingresso allorché l'operatore batte un tasto, il carattere viene ricevuto come un "treno" seriale di impulsi e convertito in un valore di 8 bit per la memorizzazione. Nella presente discussione, tale carattere di 8 bit è un carattere ASCII. Similmente, il chip è programmato per l'uscita per trasmettere un carattere ASCII quando il terminale è pronto a riceverlo. Il chip periferico in entrambi i casi fornisce la compatibilità elettrica e di temporizzazione tra il computer e il terminale. Nei sistemi basati sull'MC68020, i registri del chip di I/O sono indirizzati allo stesso modo delle locazioni di memoria.

Come il lettore vedrà nel cap. 13, la programmazione del chip periferico è un compito tedioso. In effetti, essa non è necessaria quando è richiesto questo tipo di trasferimento di I/O standard. Ovviamente, il monitor deve acquisire i caratteri dalla tastiera e rispondere ai comandi visualizzando l'informazione richiesta. Per utilizzare questa capacità del monitor dell'MVME 133, il 133BUG, basta eseguire l'istruzione TRAP #15 in un programma specificando il codice appropriato per indicare l'operazione desiderata. Altri sistemi operativi e monitor possono richiedere

un meccanismo diverso per richiedere i trasferimenti di I/O, ma i moderni sistemi di computer avranno una caratteristica equivalente. Nella maggior parte dei casi, un programma in modo utente non può programmare direttamente i chip di I/O. Quando è presente un sistema operativo, la maggior parte dei sistemi di computer richiedono che siano effettuate chiamate al sistema operativo se un programma in modo utente richiede trasferimenti di I/O. Il monitor 133BUG in questi esempi funge da sistema operativo rudimentale per il computer su piastra singola MVME133.

7.6.2 Chiamate di sistema e macro di I/O

Il monitor 133BUG dispone di un certo numero di routine, che vengono chiamate dall'istruzione TRAP # 15. Queste routine sono solitamente usate da un programmatore per facilitare l'I/O ed altre operazioni. Una è già stata incontrata negli esempi precedenti, dove l'istruzione:

```
TRAP    #15
DC.W    $0063
```

serviva per restituire il controllo al monitor al termine dell'esecuzione di un programma. L'impiego di TRAP #15 in questa maniera è talvolta indicato come una *chiamata di sistema*. La Tab. 7.10 elenca varie chiamate di sistema con i codici di TRAP #15 e descrive le rispettive funzioni.

Tab. 7.10 *Chiamate di sistema di 133BUG.*

Nome della funzione	Codice di TRAP #15	Descrizione
.INCHR	$0000	Lettura di un carattere.
.INLN	$0002	Lettura di una riga di caratteri seguita da (CR).
.READSTR	$0003	Lettura di una stringa di caratteri (fino a 254) seguiti da un (CR); la lunghezza massima della stringa è definita prima della chiamata. La stringa di caratteri produce un'eco sull'unità di visualizzazione.
.OUTCHR	$0020	Uscita di un carattere.
.WRITELN	$0024	Uscita di una stringa di caratteri (riga) seguita da (CR)(LF).
.WRITE	$0023	Uscita di una stringa di caratteri senza (CR)(LF).
.PCRLF	$0026	Uscita di (CR)(LF).
.RETURN	$0063	Ritorno al monitor.

Note alla Tab. 7.10:

1. (CR) = *Carriage-Return:* ritorno-carrello: ASCII (#0D)
   (LF) = *Line-Feed:* avanzamento-riga: ASCII (#0A)

2. Una *riga* (o linea: *line*) è una stringa di caratteri seguita da (CR); essa corrisponde di solito ad una riga di caratteri su uno schermo CRT. Una *stringa* ha una lunghezza limitata a 254 caratteri per il monitor 133BUG di MVME.

Per le funzioni d'ingresso (.INCHR, .INLN, .READSTR), dev'essere definita l'area di memoria o di buffer per il carattere, prima di effettuare la chiamata. La routine di acquisizione del carattere in input (.INCHR) può ricevere un carattere alla volta. La routine d'ingresso di riga (.INLN) può ricevere una riga di caratteri seguita da un ritorno-carrello (*Carriage Return:* CR). Le routine di uscita (.OUTCHR, .WRITELN, .WRITE) sono usate per visualizzare messaggi o dati per l'operatore. Nella Tab. 7.10 compaiono varie forme che consentono al programmatore di assegnare a suo piacimento il formato di visualizzazione su schermo. La routine .PCRLF posizionerebbe il cursore del terminale all'inizio della riga successiva sullo schermo. La routine .RETURN è un esempio di utile chiamata di sistema che non ha una relazione diretta coi trasferimenti di I/O. Essa attiva l'operazione di TRAP #15 appena discussa, col codice $0063. Il monitor 133BUG ha un certo numero di altre chiamate, non mostrate nella Tab. 7.10.

Richiesta delle chiamate di sistema. Per ogni chiamata di sistema di I/O, esclusa .PCRLF, il programmatore deve definire le locazioni da utilizzare per l'ingresso o l'uscita, come descritto dalle convenzioni di chiamata definite nella Tab. 7.11. La funzione .INCHR, per esempio, riporta un singolo carattere sullo stack di sistema. Prima della chiamata, dev'essere stata riservata una word di spazio, anche se fosse inserito soltanto un byte, poiché il puntatore dello stack di sistema dev'essere tenuto su un confine di word. Quindi il segmento di programma

```
SUBQ.L    #2,SP       ;riserva spazio
TRAP      #15         ;chiama .INCHR
DC.W      $0000
MOVE.B    (SP)+,D0    ;carattere in D0
```

alloca lo spazio, attende che un carattere sia stato acquisito ed infine trasferisce il carattere nel registro D0. Si noti che la funzione .READSTR richiede che l'indirizzo dell'area di memoria o del buffer d'ingresso sia posto sullo stack di sistema prima della chiamata. Per la stringa dev'essere stato definito il numero massimo di caratteri (byte). Il valore di questa lunghezza dev'essere specificato nel primo byte dell'area del buffer d'ingresso. Per ciascuna routine si richiede il rispetto di convenzioni simili, definite dalla Motorola per l'impiego delle chiamate di sistema. In seguito saranno forniti alcuni esempi di ciascuna chiamata.

Macrodirettive per le chiamate di sistema. Un modo per semplificare la programmazione delle funzioni di I/O e per migliorare contemporaneamente la leggibilità di un programma consiste nell'impiego delle macroistruzioni. Le macro di I/O considerate qui si limitano ad effettuare chiamate di sistema mediante il nome, come richiesto in un programma. La struttura fondamentale di una macro che svolge tale compito è la seguente:

```
SYSCALL   MACRO
          TRAP    #15
          DC.W    \1
          ENDM
```

dove \1 indica un valore da passare alla macroistruzione allorché viene assemblata. Ogni volta che si utilizza l'istruzione SYSCALL, le relative istruzioni in linguaggio assembler vengono assemblate nel programma. Quindi l'struzione

Tab. 7.11 *Convenzioni di chiamante per chiamate di sistema.*

Macro chiamata di sistema (*SYSCALL*)	**Convenzioni di chiamante**
.INCHR	Riserva una word sullo stack per il carattere.
.INLN	Inserisce sullo stack l'indirizzo del buffer d'ingresso.
.READSTR	Definisce la lunghezza massima della stringa nel buffer d'ingresso. Inserisce sullo stack l'indirizzo del buffer d'ingresso.
.OUTCHR	Inserisce il carattere sullo stack.
.WRITELN	Definisce la lunghezza della stringa nel buffer di uscita. Inserisce sullo stack l'indirizzo del buffer di uscita.
.WRITE	Definisce la lunghezza della stringa nel buffer di uscita. Inserisce sullo stack l'indirizzo del buffer di uscita.
.PCRLF	Nessuna.

Note:
1. L'espressione "Convenzioni di chiamante" significa che il programmatore deve eseguire le operazioni qui definite prima d'inviare la chiamata di sistema.
2. Lo stack di sistema è impiegato con queste SYSCALL.
3. La routine .INLN memorizza sullo stack di sistema l'indirizzo del (CR) che segue la stringa d'ingresso nel buffer d'ingresso, dopo aver letto la riga dalla tastiera.

```
SYSCALL    .INCHR
```

viene assemblata come segue:

```
TRAP       #15
DC.W       $0000
```

se .INCHR è stata eguagliata al valoe $0000, il che richiede la direttiva

```
.INCHR     EQU $0000
```

nel programma chiamante.

Esempio 7-12

Il programma di Fig. 7.13 illustra l'impiego di chiamate di sistema per svolgere varie operazioni di I/O. Le direttive di MACRO sono usate per definire la struttura chiamante seguita dai parametri per la particolare funzione. Quando viene eseguito, per prima cosa il programma visualizza sul terminale dell'operatore il testo ASCII all'etichetta LINE1. Prima di visualizzare i caratteri di uscita definiti da LINE1, LINE2, LINE3, LINE4 e LINE5, ogni indirizzo viene posto sullo stack di sistema al posto giusto nel programma, mediante l'istruzione PEA (*Push Effective Address:* inserisci indirizzo effettivo). [L'istruzione PEA sarà discussa nel cap. 9.] Le prime tre chiamate di sistema vengono espanse dalla direttiva OPT MEX (*Macro EXpansion:* espansione di macro), per mostrare le istruzioni del linguaggio assembler che corrispondono alle macro. Le rimanenti vengono eliminate dal listato in linguaggio assembler usando la direttiva OPT NOMEX (*NO Macro EXpansion:* nessuna espansione di macro). Una stampa causata da una routine di tracciamento nella Fig. 7.14 illustra lo schermo di visualizzazione, così com'è visto dall'operatore. Dopo che è stato visualizzato il testo corrispondente a LINE2, l'operatore digita un carattere seguito da (CR). Questo carattere non è visualizzato sullo schermo mentre viene digitato, ma viene memorizzato nello stack di sistema e trasferito al registro D0. Esso viene rivisualizzato o "echeggiato" sullo schermo dopo il testo LINE3 dalla chiamata di sistema .OUTCHR.

Dopodiché, viene ricevuta e visualizzata una stringa di caratteri dopo il testo di LINE4. La lunghezza massima della stringa è definita come 254 caratteri nel primo byte del buffer d'ingresso. Un'area di 256 byte è riservata come INBUF per la stringa d'ingresso nella sezione di dati del programma. La chiamata di sistema .READSTR legge ed echeggia la stringa di caratteri dopo che l'operatore l'ha digitata, facendola seguire da un ritorno-carrello. La chiamata lascia la lunghezza effettiva della stringa nel primo byte del buffer INBUF. Questo buffer serve anche come buffer di uscita per la successiva chiamata di sistema a .WRITELN per echeggiare nuovamente la stringa.

Le chiamate a .WRITELN visualizzano una riga o stringa di caratteri seguita da (CR) e (LF). Per contro, una chiamata di sistema a .WRITE lascerebbe il cursore dell'unità CRT in una posizione immediatamente successiva a quella dell'ultimo carattere visualizzato, a meno che i caratteri di ritorno-carrello (CR) e di avanzamento-riga (LF) non facessero parte della stringa, come avveniva nel testo di LINE5. Il programmatore può quindi assegnare il formato di visualizzazione desiderato, trasmettendo al terminale i caratteri di controllo del formato. Per esempio, quando viene emesso in uscita il testo di LINE5, sono anche inviati i caratteri (CR) e (LF), i cui codici ASCII sono rispettivamente $0D e $0A. Selezionando i caratteri di controllo del formato appropriato, il programmatore può causare azioni quali la "cancellazione" dello schermo o la tabulazione di qualsiasi risultato visualizzato. Il particolare codice ASCII per tale impostazione dipende dal formato richiesto dal terminale dell'operatore.

```
                              1.           TTL      FIGURA 7.13
                              2.           LLEN     100
00040000                      3.           ORG      $40000
                              4.           OPT      MEX                ;ESPANDE LE MACRO
                              5.   *
                              6.   *     CHIAMATA DI SISTEMA DI I/O ("SYSCALL")
                              7.   *
                              8.m  SYSCALL  MACRO                      ;MACRO = SYSCALL
                              9.m           TRAP     #15
                             10.m           DC.W     \1                ;PARAMETRO
                             11.m           ENDM
                             12.   *
                             13.   *     PARAMETRI DELLA SYSCALL DI I/O
                             14.   *
      = 00000000             15.   .INCHR   EQU      $0000             ;INPUT CARATTERE
      = 00000002             16.   .INLN    EQU      $0002             ;INPUT RIGA
      = 00000003             17.   .READSTR EQU      $0003             ;INPUT STRINGA
      = 00000020             18.   .OUTCHR  EQU      $0020             ;OUTPUT CARATTERE
      = 00000024             19.   .WRITELN EQU      $0024             ;OUTPUT RIGA CON
                             20.   *                                   ;  (CR),(LF)
      = 00000023             21.   .WRITE   EQU      $0023             ;OUTPUT STRINGA
      = 00000026             22.   .PCRLF   EQU      $0026             ;(CR),(LF)
      = 00000063             23.   .RETURN  EQU      $0063             ;RITORNA AL MONITOR
                             24.   *
                             25.   *     TEST DI VARIE CHIAMATE
                             26.   *
                             27.   *
                             28.   *       INVIA IN USCITA IL MESSAGGIO "TEST OF I/O SYSCALLS"
                             29.   *
00040000  4879 00040072      30.   STARTIO  PEA      LINE1             ;INDIRIZZO DEL MESSAGGIO:
                             31.   *                                   ;  RIGA 1 (LINE 1)
                             32.c           SYSCALL .WRITELN           ;SCRIVE SULLO SCHERMO
00040006  4E4F               33.1           TRAP     #15
00040008  0024               34.1           DC.W     .WRITELN             ;PARAMETRO
                             35.   *
                             36.   *        ACQUISICE UN CARATTERE E LO ECHEGGIA SULLO SCHERMO
                             37.   *
0004000A  4879 00040088      38.            PEA      LINE2             ;RICHIESTA DI UN CARATTERE
                             39.c           SYSCALL .WRITELN
00040010  4E4F               40.1           TRAP     #15
00040012  0024               41.1           DC.W     .WRITELN             ;PARAMETRO
00040014  558F               42.            SUBQ.L   #2,SP             ;SALVA SPAZIO
                             43.c           SYSCALL .INCHR             ;INPUT DI CARATTERE
00040016  4E4F               44.1           TRAP     #15
00040018  0000               45.1           DC.W     .INCHR            ;PARAMETRO
0004001A  101F               46.            MOVE.B   (SP)+,D0          ;CARATTERE IN (D0)
                             47.   *
                             48.            OPT      NOMEX             ;SOPPRIME L'ESPANSIONE
                             49.   *
0004001C  4879 0004009B      50.            PEA      LINE3             ;ECO
                             51.c           SYSCALL .WRITE
00040026  1F00               54.            MOVE.B   D0,-(SP)
                             55.c           SYSCALL .OUTCHR
                             58.c           SYSCALL .PCRLF             ;RIGA SUCCESSIVA
                             61.
```

Fig. 7.13 *Esempi di chiamate di I/O di sistema.*

```
Quelo ...A68K K6.2J2  June 29, 1987  ...Run on Jun  4, 1988  13:44:08...Page   2
f7_13.ltx , f7_13.prn , f7_13.sym = f7_13.a68
  ...FIGURA 7.13

                               63.  *
                               64.  *       INPUT ED ECO DI UNA STRINGA DI CARATTERI
                               65.  *
                               66.c         SYSCALL .PCRLF          ;SALTA UNA RIGA
00040034  4879 000400AB        69.          PEA     LINE4           ;RICHIESTA DI STRINGA
                               70.c         SYSCALL .WRITELN
0004003E  13FC 00FE            73.          MOVE.B  #254,INBUF      ;LUNGHEZZA MASSIMA
          00040100
00040046  4879 00040100        74.          PEA     INBUF           ;INDIRIZZO DI BUFFER
                               75.c         SYSCALL .READSTR        ;LEGGE LA STRINGA
                               78.c         SYSCALL .PCRLF          ;  E LA ECHEGGIA
00040054  1039 00040100        81.          MOVE.B  INBUF,D0        ;LUNGHEZZA EFFETTIVA
                               82.  *
0004005A  4879 000400C9        83.          PEA     LINE5           ;RIECHEGGIA LA STRINGA
                               84.c         SYSCALL .WRITE
                               87.  *
00040064  4879 00040100        88.          PEA     INBUF
                               89.c         SYSCALL .WRITELN
                               92.  *
                               93.c         SYSCALL .RETURN         ;RITORNA
                               96.  *
                               97.  *       MESSAGGI E DATI
                               98.  *
00040072  15 20 54 45 53       99.  LINE1   DC.B    21,' TEST OF I/O SYSCALL'
          54 20 4F 46 20
          49 2F 4F 20 53
          59 53 43 41 4C
          4C 53
00040088  12 20 49 4E 50      100.  LINE2   DC.B    18,' INPUT A CHARACTER'
          55 54 20 41 20
          43 48 51 52 41
          43 54 45 52
0004009B  0F 20 43 48 41      101.  LINE3   DC.B    15,' CHARACTER IS'
          52 41 43 54 45
          52 20 49 53 20
          20
000400AB  1D 20 49 4E 50      102.  LINE4   DC.B    29,' INPUT A STRING OF CHARACTERS'
          55 54 20 41 20
          53 54 52 49 4E
          47 20 4F 46 20
          43 48 41 52 41
          43 54 45 52 53
000400C9  0C 20 53 54 52      103.  LINE5   DC.B    12,' STRING IS',$0D,$0A
          49 4E 47 20 49
          53 0D 0A
                              104.  *
00040100                      105.          ORG $40100
00040100  <100>               106.  INBUF   DS.B    256             ;BUFFER DI INPUT
                              107.  *
                              108.* ...M68K K2.1M  June 29, 1987  ...Run on Jun  4, 1988  13:43:48
                              109.          END
```

Figura 7.13 *(continuazione)*

```
133Bug>RM PC
PC    =00000000 ? 40000.
133Bug>GO
Effective address: 00040000
 TEST OF I/O SYSCALLS
 INPUT A CHARACTER
 CHARACTER IS  5

 INPUT A STRING OF CHARACTERS
 TOM HARMAN
 STRING IS
 TOM HARMAN
```

Fig. 7.14 *Traccia del programma di Fig. 7.13.*

7.6.3 Conversioni di dati – le istruzioni PACK e UNPACK.

Le rappresentazioni standard di dati per l'MC68020 includono quelle binarie e BCD per gli interi e i codici ASCII per i caratteri. Per l'ingresso e l'uscita, il codice ASCII è in generale il codice impiegato per i trasferimenti di dati tra il sistema del computer e i dispositivi periferici come le stampanti parallele o i terminali CRT. Le conversioni tra queste rappresentazioni sono quindi frequentemente richieste, poiché l'elaborazione aritmetica richiede valori binari o BCD nella memoria.

La Fig. 7.15(a) mostra i passi del procedimento tipico per convertire in una rappresentazione binaria un numero decimale espresso in ASCII. Dapprima i caratteri ASCII delle cifre decimali vengono convertiti in numeri binari nell'intervallo 0-9. I 4 bit di ciascuna cifra sono anche il valore BCD nella memoria. Dopodiché la stringa di cifre BCD viene convertita in un numero binario. Questa conversione tiene conto del valore posizionale di ciascuna cifra BCD. Per esempio, la stringa ASCII '123' come valore d'ingresso è registrata nella memoria come $31, $32 e $33. Questi valori vengono convertiti in tre cifre BCD, 1, 2 e 3, e poi nel valore binario 01111011.

L'uscita di valori binari che devono essere stampati come numeri decimali richiede la conversione opposta da binario a BCD e poi ad ASCII. Naturalmente, il valore binario nella memoria potrebbe essere stampato in binario, esadecimale, o in un altro codice. Il numero binario nell'esempio di ingresso appena visto ha il valore esadecimale 7B. Questo potrebbe essere emesso in uscita nella forma ASCII $37, $42. La Fig. 7.15(b) mostra gli equivalenti ASCII delle cifre decimali ed esadecimali.

Conversioni tra ASCII e BCD mediante le istruzioni PACK e UNPACK. La Tab. 7.12 mostra la sintassi e le operazioni delle istruzioni PACK e UNPACK. L'istruzione PACK viene usata principalmente per convertire l'operando di sorgente da una stringa ASCII di due byte (due caratteri) in un BCD impaccato. Il risultato può essere usato direttamente dalle istruzioni aritmetiche BCD dell'MC68020. Poiché l'istruzione di *impaccamento di BCD*

```
PACK        D2,D3,#$0000
```

acquisisce i 4 bit meno significativi di ciascun byte della word di sorgente e li trasferisce nel byte di destinazione, la conversione da ASCII BCD è completa se D2 contiene due numeri ASCII rappresentati come 3X3Y, dove X e Y sono le cifre decimali.

L'istruzione di *disimpaccamento di BCD*

```
UNPK        D2,D3,#$3030
```

Fig. 7.15
Conversioni di valori di dati.

Cifre decimali 'NNNN' in ASCII → Conversione da ASCII a BCD → Conversione da BCD a binario

(a) *Conversione di dati d'ingresso.*

Significato	Rappresentazione binaria nella memoria	Per convertire in ASCII	ASCII nella memoria
Cifre decimali 0-9 (BCD)	0000 0000 0000 0001 ⋮ 0000 1001	Sommare $0011\ 0000_2$ ($30)	$30-$39
Cifre esadecimali 0-9	0000 0000 0000 0001 ⋮ 0000 1001	Sommare $30	$30-$39
Cifre esadecimali A-F	0000 1010 0000 1011 ⋮ 0000 1111	Sommare $37	$41-$46

(b) *Da decimale o esadecimale ad ASCII per uscita.*

converte due cifre BCD memorizzate come 00XY in D2 nel valore 3X3Y in D3. Se fosse usato un valore di dati diverso da $3030, le cifre BCD potrebbero essere convertite in una forma diversa da quella ASCII. Per esempio, usando $0000 come valore di dati, un valore BCD di due cifre viene convertito in un BCD disimpaccato. Questo risultato potrebbe essere utilizzato per le operazioni di moltiplicazione o di divisione di BCD, in cui sono richiesti valori BCD non impaccati.

Tab. 7.12 *Le istruzioni PACK e UNPK.*

Istruzione	Operazione
PACK −(Am), −(An),# <dato>	<sorg> 15 11 8 7 3 0 (← numero di bit): [ ] [D_1] [ ] [D_0]
	+
PACK <Dm>,<Dn>,# <dato>	<dato> 15 11 8 7 3 0: [ ] [ ] [ ] [ ]
	<dest> [D_1'] [D_0'] 7 4 3 0
UNPK −(Am), −(Am),# <dato>	<sorg> 15 8 7 4 3 0 (← numero di bit): [ ] [ ] [D_1'] [D_0']
	+ +
UNPK <Dm>,<Dn>,# <dato>	<dato> 15 11 8 7 4 3 0: [ ] [ ] [ ] [ ]
	<dest> 15 11 8 7 43 0: [ ] [D_1] [ ] [D_0]

Note:

1. <dato> è un intero con segno di 16 bit.
2. Quando gli operandi sono contenuti nella memoria, il byte più significativo di <sorg> deve trovarsi all'indirizzo di memoria inferiore.
3. Am e An denotano registri d'indirizzo arbitrari. Dm e Dn indicano registri di dati arbitrari.

Esempio 7-13

Il programma della Fig. 7.16 converte una stringa di valori ASCII in un formato BCD impaccato. Si potrebbe usare una chiamata di sistema come la .READSTR (discussa in precedenza) per leggere la stringa da una tastiera. All'atto dell'inserimento, il registro (A0) punta all'inizio della stringa. Il suo byte iniziale dev'essere la lunghezza della stringa (numero di cifre); ciascun byte successivo, da (A0)+1 fino ad (A0)+N dovrebbe contenere un valido carattere ASCII rappresentativo di una cifra BCD. Se quest'ultima non fosse nell'intervallo 0-9, allora sarebbe fornita un'indicazione di errore. Prima che il programma sia eseguito, il valore in A1 dev'essere uguale all'ultimo indirizzo + 1 dell'array che conterrà le cifre BCD impaccate. Quindi la struttura nella memoria è quella rappresentata nella Fig. 7.17.

```
00040300                 1.                TTL      FIGURA 7.16
                         2.                LLEN     100
00040300                 3.                ORG      $40300
    # 00000063           4.        .RETURN EQU      $0063
                         5.                XDEF     PACKBCD
                         6.        #
                         7.        #       CONVERSIONE DA ASCII A BCD IMPACCATO
                         8.        #
                         9.        #       INPUT:   (A0.L) = INDIRIZZO INIZIALE DELLA STRINGA
                        10.        #                         DI NUMERI ASCII DI LUNGHEZZA N.
                        11.        #                         IL 1^ BYTE DEVE CONTENERE N.
                        12.        #                         N DEV'ESSERE PARI.
                        13.        #
                        14.        #                (A1.L) = IN ENTRATA, (A1) DOVREBBE PUNTARE 1 BYTE
                        15.        #                         DOPO (+1) L'ARRAY PER I RISULTATI
                        16.        #                         IN FORMA BCD IMPACCATA.
                        17.        #
                        18.        #       OUTPUT:  (A1.L) = INDIRIZZO DI ARRAY DI CIFRE
                        19.        #                         BCD IMPACCATE.
                        20.        #                (D1.W) = 0      SE LA CONVERSIONE E' CORRETTA
                        21.        #                       = $FFFF  SE IL VALORE ASCII E' SCORRETTO
                        22.        #                                O SE LA LUNGHEZZA E' ZERO.
                        23.        #
00040300  48E7 3080     24.        PACKBCD MOVEM.L D2/D3/A0,-(SP)   ;SALVA I REGISTRI
00040304  4241          25.                CLR.W    D1              ;IMPOSTA PER STATO CORRETTO
00040306  1218          26.                MOVE.B   (A0)+,D1        ;LUNGHEZZA N IN D1
00040308  1601          27.                MOVE.B   D1,D3           ;  E IN D3
0004030A  6700 0024     28.                BEQ      ERROR           ;ERRORE SE LA LUNGHEZZA
                        29.        #                                ;E' ZERO
0004030E  1418          30.        CHECK   MOVE.B   (A0)+,D2        ;CARATTERE IN D2
00040310  0C02 0030     31.                CMPI.B   #$30,D2         ;SE CARATTERE < $30
00040314  6D00 001A     32.                BLT      ERROR           ;  ALLORA NON VALIDO
00040318  0C02 0039     33.                CMPI.B   #$30,D2         ;SE CARATTERE > $39
0004031C  6E00 0012     34.                BGT      ERROR           ;  ALLORA NON VALIDO
00040320  5301          35.                SUBQ.B   #1,D1           ;ALTRIMENTI, ESAMINA
00040322  66 EA         36.                BNE      CHECK           ;  IL CARATTERE SUCCESSIVO
                        37.        #
                        38.        #       CONVERTE DUE CARATTERI ASCII IN CBD IMPACCATO
                        39.        #
00040324  8348 0000     40.        PACK    PACK     -(A0),-(A1),#$0000
00040328  5503          41.                SUBQ.B   #2,D3           ;CONVERTE 2 CIFRE FINO
0004032A  66 F8         42.                BNE      PACK            ; A RAGGIUNGERE N CIFRE
0004032C  6000 0006     43.                BRA      FINE
                        44.        #
00040330  323C FFFF     45.        ERROR   MOVE.W   #$FFFF,D1       ;IMPOSTA PER STATO DI ERRORE
                        46.        #
00040334  4CDF 010C     47.        FINE    MOVEM.L (SP)+,D2/D3/A0   ;RIPRISTINA I REGISTRI
00040338  4E4F          48.                TRAP     #15             ;  E RITORNA
0004033A  0063          49.                DC.W     .RETURN
0004033C                50.                END
```

Fig. 7.16 *Esempio di conversione da ASCII a BCD impaccato.*

INGRESSO		USCITA	
(A0) →	n. di byte		
	$'D_{N-1}'$		
	$'D_{N-2}'$	(A) →	$D_{N-1}\ D_{N-2}$
	⋮		⋮
	$'D_1'$		$D_3\ D_2$
	$'D_0'$		$D_1\ D_0$
	PRIMA		DOPO

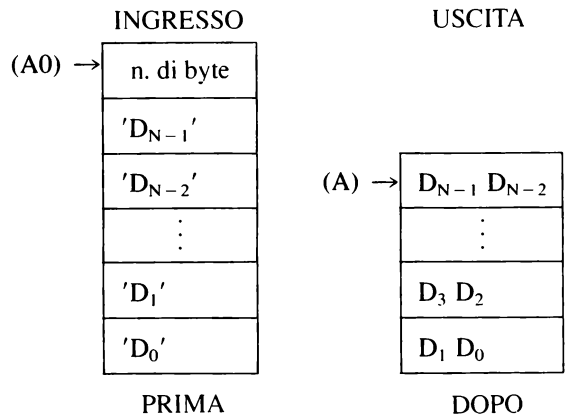



Fig. 7.17
Struttura della memoria.

Nella Fig. 7.17, 'D$_i$' indica l'equivalente ASCII della cifra D$_i$, con i = 0, 1, ..., N − 1. Il programma vincola N, il numero di cifre, ad essere un numero pari. Quindi l'array dei dati d'ingresso ha N + 1 byte, mentre l'array di BCD impaccati ha N/2 elementi.

Al termine del programma, D1 indica i risultati. Se (D1)[15:0] è zero, la conversione è riuscita. Altrimenti, la word meno significativa di D1 conterrà un indicatore di errore −1.

Esempio 7-14

La Fig. 7.18 mostra una subroutine per determinare il valore ASCII o BCD di una cifra esadecimale memorizzata nei 4 bit meno significativi di D1. Se (D2)[7:0] contiene un valore 0 all'inserimento, il codice ASCII viene posto nel byte meno significativo di D2. Il valore viene trovato indirizzando la tabella ASCTAB ed effettuando l'indicizzazione in base alla cifra esadecimale in D1. Se (D2)[7:0] vale 1 all'inserimento, la cifra BCD che corrisponde alla cifra esadecimale viene posta in D2. Il valore è ottenuto dalla tabella BCDTAB. Questo metodo di ricerca tabellare è un'alternativa per convertire valori da una forma ad un'altra.

```
                         1.          TTL      FIGURA 7.18
                         2.          LLEN     100
00040800                 3.          ORG      $40800
                         4.          XDEF     CONHAS
                         5.  *
                         6.  *       CONVERSIONE DA ESADECIMALE A BCD/ASCII
                         7.  *
                         8.  *       INPUT:   (D2.B) = 0      : ASCII RICHIESTO
                         9.  *                       NON 0 : BCD RICHIESTO
                        10.  *                (D1.W) = LA CIFRA ESADECIMALE DA CONVERTIRE
                        11.  *                         DEV'ESSERE VALIDA: 0-F
                        12.  *
                        13.  *       OUTPUT:  (D2.B) = BCD O ASCII
                        14.  *
                        15.  *
                        16.  *
00040800  2F09          17.  CONHAS  MOVE.L   A1,-(SP)                 ;SALVA I REGISTRI
00040802  227C 0004081C 18.          MOVE.L   #ASCTAB,A1               ;IPOTESI: RICHIESTO ASCII
00040808  4A02          19.          TST.B    D2                       ;CONTROLLA LA RICHIESTA
0004080A  6700 0008     20.          BEQ      INDEX                    ;  SE RICHIESTO BCD
0004080E  227C 0004082C 21.          MOVE.L   BCDTAB,A1                ;  ALLORA TABELLA DI BCD
                        22.  *
00040814  1431 1000     23.  INDEX   MOVE.B   0(A1,D1.W),D2            ;CERCA IL VALORE
00040818  225F          24.          MOVE.L   (SP)+,A1                 ;RIPRISTINA I REGISTRI
0004081A  4E75          25.          RTS
0004081C  30 31 32 33 34 26.  ASCTAB DC.B     '0123456789ABCDEF'
          35 36 37 38 39
          41 42 43 44 45
          46
0004082C  00 01 02 03 04 27.  BCDTAB DC.B     0,1,2,3,4,5,6
          05 06
00040833  07 08 09 10 11 28.          DC.B     7,8,9,$10,$11
00040838  12 13 14 15    29.          DC.B     $12,$13,$14,$15
0004083C                 30.          END
```

Fig. 7.18 *Ricerche tabellari per la conversione da esadecimale a BCD/ASCII.*

Esempio 7-15

Se una stringa di cifre decimali è memorizzata come $D_{n-1}D_{n-2}\cdots D_0$ in byte separati nella memoria, la conversione in binario si consegue facilmente poiché il valore numerico è:

$$(\cdots(D_{n-1} \times 10 + D_{n-2}) \times 10 + \cdots + D_1 \times 10) + D_0$$

come spiegato nel cap. 3. La somma binaria è formata calcolando i termini entro le parentesi mediante l'aritmetica binaria e sommando ciascun termine nel totale. La Fig. 7.19 mostra una subroutine per effettuare questa conversione per un valore decimale BCD di otto cifre. Si presuppone che, all'atto dell'inserimento, le cifre decimali siano memorizzate con giustificazione a destra nelle locazioni di byte indirizzate da A1. Se non si presenta alcuna condizione di fuori-intervallo, il valore binario sarà riportato in D1.

```
                         1.          TTL      FIGURA 7.19
                         2.          LLEN     100
00040600                 3.          ORG      $40600
                         4.          XDEF     BCDBN
                         5.  *
                         6.  *       CONVERSIONE DA BCD A BINARIO
                         7.  *
                         8.  *       INPUT:  (D2.W) = NUMERO DI CIFRE NEL VALORE BCD
                         9.  *               (A1.L) = INDIRIZZO DELLA CIFRA BCD
                        10.  *                        PIU' SIGNIFICATIVA
                        11.  *
                        12.  *       OUTPUT: (D1.L) = VALORE BINARIO
                        13.  *               (D4.W) = 0     : RIVELATO ERRORE
                        14.  *                        NON 0 : CONVERSIONE RIUSCITA
                        15.  *
                        16.  *       NOTE:
                        17.  *       1.  LE CIFRE BCD SONO MEMORIZZATE UNA/BYTE
                        18.  *           E DEVONO ESSERE VALIDE. IL LIMITE E' DI 8 CIFRE.
                        19.  *       2.  SONO AMMESSI SOLTANTO NUMERI BCD POSITIVI.
                        20.  *
                        21.  *
                        22.  *
00040600  48E7 3440     23.  BCDBN   MOVEM.L D2/D3/D5/A1,-(SP)     ;SALVA I REGISTRI
00040604  4281          24.          CLR.L   D1                    ;VALORE := 0
00040606  4244          25.          CLR.W   D4                    ;PREFISSA STATO PER ERRORE
00040608  4285          26.          CLR.L   D5                    ;AZZERA ACCUMULATORE
0004060A  4A42          27.          TST.W   D2                    ;SE LA LUNGHEZZA E' ZERO
0004060C  6700 002E     28.          BEQ     FINE                  ;  ALLORA ESCE CON ERRORE
00040610  0C42 0008     29.          CMPI.W  #8,D2                 ;SE LUNGHEZZA > 8
00060014  6E00 0026     30.          BGT     FINE                  ;  ALLORA ESCE CON ERRORE
                        31.
                        32.  *
00040618  1A19          33.  LOOP    MOVE.B  (A1)+,D5              ;CIFRA IN D5
0004061A  D285          34.          ADD.L   D5,D1                 ;SOMMA ALLA SERIE
0004061E  6700 0018     35.          SUBQ.W  #1,D2                 ;DECREMENTA IL CONTATORE
                        36.          BEQ     SUCCESS               ;SE TERMINATO (ULTIMA CIFRA)
                        37.  *                                     ;  ALLORA ESCE
                        38.  *
                        39.  *       MOLTIPLICA PER 10 COME 10X = (2X) * 4 + 2X
                        40.  *
00040622  E389          41.          LSL.L   #1,D1                 ;MOLTIPLICA PER 2
00040624  6B00 0016     42.          BMI     FINE                  ;SE OVERFLOW, ESCE
                        43.  *
00040628  2601          44.          MOVE.L  D1,D3                 ;SALVA IL PRODOTTO
0004062A  E589          45.          LSL.L   #2,D1                 ;MOLTIPL. PER 4 (VALORE X 8)
0004062C  6B00 000E     46.          BMI     FINE                  ;SE OVERFLOW, ESCE
                        47.  *
00040630  D283          48.          ADD.L   D3,D1                 ;VALORE = VALORE * 10
00040632  6B00 0008     49.          BMI     FINE                  ;SE OVERFLOW, ESCE
                        50.  *
00040636  60   E0       51.          BRA     LOOP                  ;ELABORA LA CIFRA SUCC.
00040638  383C 0001     52.  SUCCESS MOVE.W  #1,D4                 ;IMPOSTA STATO PER SUCCESSO
0004063C  4CDF 022C     53.  FINE    MOVEM.L (SP)+,D2/D3/D5/A1     ;RIPRISTINA I REGISTRI
00040640  4E75          54.          RTS
00040642                55.          END
```

Fig. 7.19 *Routine di conversione da BCD a binario.*

Nel programma, la moltiplicazione per 10 è ottenuta facendo scorrere i bit verso sinistra (moltiplicazione per 2) e sommando, anziché utilizzando l'istruzione MULU. Questo codice potrebbe essere sostituito dall'istruzione di moltiplicazione e da un test di overflow per un numero maggiore di 99999999. Si potrebbe verificare un errore soltanto se il valore decimale di otto cifre da convertire non è un valido numero BCD. Nel programma, così com'è scritto, si presenta una condizione di overflow per un numero binario maggiore di 2^{31}, poiché viene impiegata l'istruzione BMI.

ESERCIZI

7.6.1 Si scriva una subroutine per convertire una stringa ASCII in una rappresentazione binaria con segno, quando l'intervallo dell'ingresso può estendersi fino a 8 cifre decimali con segno.

7.6.2 Si scriva una subroutine per convertire in ASCII una stringa di cifre binarie.

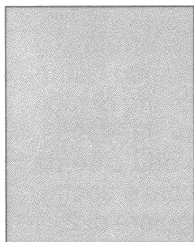
7.6.3 Per convertire da binario in decimale, è possibile seguire il procedimento mostrato in questo paragrafo, usando l'espansione decimale per il numero da convertire. Dividendo ripetutamente il numero per 10, si otterranno le cifre decimali, come resti, in ordine ascendente. Si scriva una subroutine per convertire un intero senza segno di 16 bit nel valore BCD equivalente.

7.6.4 Si scriva una routine per convertire in BCD una stringa di cifre ASCII quando la stringa d'ingresso contiene un punto decimale ($2E). Si determini il numero di cifre e si lasci il fattore di scala in un registro; cioè, si determini il numero di posizioni decimali e si memorizzi il numero BCD come un intero.

7.6.5 Si modifichi il programma dell'esempio 7.15 per convertire in binario una stringa di valori decimali ASCII. Si limiti la stringa a numeri positivi non più lunghi i 8 cifre.

7.6.6 Si scrivano le istruzioni per effettuare le seguenti conversioni:

(a) Da BCD impaccato a EBCDIC (*Extended BCD Interchange Code:* codice di scambio BCD esteso, è un codice simile a quello ASCII, la cui diffusione è dovuta al suo impiego nei grandi sistemi di computer IBM). Il lettore potrebbe consultare l'appropriata letteratura tecnica dell'IBM per i codici EBCDIC.
(b) Da EBCDIC a BCD impaccato.
(c) Dalle cifre esadecimali $0-$F ad ASCII.

7.6.7 Per il sistema di computer utilizzato, si crei un insieme di programmi di "servizio" (*utilities*) che svolgano le seguenti funzioni:

(a) Acquisire valori ASCII decimali e convertirli in BCD impaccati o disimpaccati.
(b) Acquisire valori ASCII in decimale e convertirli in binario. Si considerino interi sia con segno che senza segno.
(c) Presentare in uscita i risultati decimali in ASCII dopo la conversione da BCD o binario.
(d) Presentare in uscita i risultati esadecimali in ASCII per visualizzare un *dump* (letteralmente: "scaricamento") di regioni selezionate della memoria.
(e) Creare una libreria di macroistruzioni di I/O e di routine di conversione per svolgere le precedenti operazioni.

CAPITOLO 8

OPERAZIONI LOGICHE E DI BIT

In questo capitolo sono presentate quattro nuove categorie di istruzioni dell'MC68020: istruzioni logiche, istruzioni di scorrimento e rotazione, istruzioni di manipolazione di bit e istruzioni di campo di bit. Le *operazioni logiche* trattano un operando come una raccolta di variabili logiche distinte. Questa categoria include le istruzioni AND, OR, EOR (OR esclusivo), e NOT. La seconda categoria comprende le istruzioni ASL, ASR, LSL e LSR per lo *scorrimento* dei bit entro un operando. Sono disponibili istruzioni di scorrimento sia aritmetico che logico. Le istruzioni ROL, ROR, ROXL e ROXR fanno ruotare in maniera ciclica i bit entro un operando.

Le istruzioni per la *manipolazione di bit* formano una categoria separata di istruzioni per l'MC68020. Sono disponibili istruzioni distinte per il test, per l'assegnazione del valore {1} o del valore {0} e per la modifica di un singolo bit entro un operando. In ordine, esse hanno i codici mnemonici BTST, BSET, BLCR e BCHG. Due altre istruzioni mostrano il risultato di un test condizionale tramite la modifica di una variabile di indicatore o "flag": si tratta delle istruzioni Scc (*Set according to condition:* assegna il valore in base alla condizione) e TAS (*Test And Set:* esamina e imposta, cioè: esegui il test e assegna il valore).

Quando un'applicazione richiede la manipolazione di stringhe di bit che non sono necessariamente composte da 8, 16 o 32 bit, si rivelano utili le istruzioni di *campo di bit* dell'MC68020. Queste istruzioni operano su stringhe di bit di lunghezza variabile, denominate "campi di bit". Nell'ultimo paragrafo di questo capitolo, saranno descritte le istruzioni di campo di bit e le relative applicazioni.

Le istruzioni logiche, di manipolazione di bit e di campo di bit sono di considerevole utilità nel trattamento di singoli bit o di una sequenza di bit contigui. I bit d'interesse spesso fanno parte di una maggiore struttura di dati, come una word di stato, una stringa di caratteri o una mappa di bit. Quando la struttura di dati non è comodamente suddivisa in entità di byte, word o longword, le istruzioni presentate in questo capitolo possono semplificare la programmazione e ridurre il tempo di esecuzione di routine che devono manipolare variabili logiche o stringhe di bit.

8.1 OPERAZIONI LOGICHE

In alcune applicazioni, è comodo trattare ciascun bit in un operando come una variabile logica individuale. Per esempio, il registro dei codici di condizione contiene cinque bit indipendenti, che possono essere esaminati singolarmente. Ogni variabile logica ha soltanto due stati possibili, che sono definiti diversamente, a seconda dell'applicazione, come VERO o FALSO, ON o OFF, {1} o {0}, tra le varie possibilità. Pertanto, una word di computer a *m* bit contiene *m* variabili logiche. Nell'MC68020 le istruzioni logiche possono operare simultaneamente su 8, 16 o 32 variabili di questo tipo.

Se *x* e *y* sono considerate variabili logiche, le tavole della verità nella Tab. 8.1 definiscono le operazioni che corrispondono all'aritmetica logica dell'MC68020. Una raccolta di *m* variabili logiche è scritta in notazione posizionale

$$(x_{m-j}\, x_{m-2} \ldots x_0)$$

come se fosse memorizzata in una word di *m* bit. Questa è definita una *m*-tupla di variabili. Per esempio, l'istruzione dell'MC68020

```
NOT.W      X
```

effettuerà il complemento di ciascun bit di un operando che contiene 16 variabili logiche, nella locazione di memoria indirizzata da X. Le altre istruzioni dell'MC68020 per le operazioni logiche svolgono le relative operazioni tra le variabili logiche nella locazione di sorgente e le variabili logiche nella destinazione. Il risultato viene memorizzato nella locazione di destinazione. Quindi l'operazione:

```
AND.W      D1,D2
```

lascia nella word meno significativa di D2 il risultato dell'operazione tra 16 variabili in D1 e 16 variabili in D2. L'operazione eseguita è

$$(D2)[15:0] \leftarrow (D2)[15:0] \text{ AND } (D1)[15:0]$$

Tab. 8.1 *Risultati di operazioni logiche.*

x	*y*	NOT *x*	*x* AND *y*	*x* OR *y*	*x* EOR *y*
0	0	1	0	0	0
0	1	1	0	1	1
1	0	0	0	1	1
1	1	0	1	1	0

Nota:
x e *y* sono variabili logiche. I risultati di ciascuna operazione sono definiti dalla tavola della "verità" per l'operazione. Ad esempio, (*x* OR *y*) è vera, cioè {1}, se *x* o *y* o entrambe valgono {1}.

Le istruzioni logiche sono elencate nella Tab. 8.2, che mostra la sintassi di assemblatore e le modalità d'indirizzamento per ciascuna istruzione. Le istruzioni con suffisso "I" ammettono soltanto un valore immediato per l'operando di sorgente. Nessuna delle istruzioni elencate opera su registri d'indirizzo. Inoltre, non è possibile alcuna operazione da memoria a memoria con le istruzioni AND, EOR e OR. I bit C e V del registro dei codici di condizione sono sempre azzerati dopo qualsiasi operazione logica, mentre i bit N e Z sono impostati in base al risultato. Il bit Z sarebbe posto a {1} se tutti i bit del risultato sono zeri. Il bit N è posto a {1} se il bit più significativo del risultato è {1}.

Le istruzioni AND e OR ammettono le medesime modalità d'indirizzamento per i loro operandi. Per l'una o l'altra istruzione, se la destinazione è un registro di dati, l'operando di sorgente è indirizzato da una qualsiasi modalità di dati. Ciò rende possibili tutti i modi d'indirizzamento per la sorgente, tranne quello diretto di registro d'indirizzo. In alternativa, se Dn contiene l'operando di sorgente, la destinazione dev'essere indirizzata da una modalità alterabile di memoria. Quindi per la destinazione sono proibiti soltanto il modo d'indirizzamento diretto di registro e quello relativo al contatore di programma.

Tab. 8.2 *Istruzioni per le operazioni logiche.*

Sintassi		Modalità d'indirizzamento	
		Sorgente	**Destinazione**
AND logico			
AND.<l>	<EA>,<Dn>	Dati	<Dn>
AND.<l>	<Dn>,<EA>	<Dn>	Alterabile di memoria
ANDI.<l>	#<d>,<EA>	<d>	Alterabile di dati
OR logico			
OR.<l>	<EA>,<Dn>	Dati	<Dn>
OR.<l>	<Dn>,<EA>	<Dn>	Alterabile di memoria
ORI.<l>	#<d>,<EA>	<d>	Alterabile di dati
OR esclusivo			
EOR.<l>	<Dn>,<EA>	<Dn>	Alterabile di dati
EORI.<l>	#<d>,<EA>	<d>	Alterabile di dati
NOT			
NOT.<l>	<EA>	---	Alterabile di dati

Note:
1. <l> denota B, W o L.
2. <d> è una variabile logica di 8, 16 o 32 bit come valore immediato.
3. I bit C e V di codici di condizione sono sempre azzerati; N e Z sono impostati in base al risultato.
4. La locazione di destinazione viene modificata in base al risultato.

Le istruzioni ANDI e ORI impiegano un valore immediato come operando di sorgente e una qualsiasi locazione alterabile di dati per l'operando di destinazione. Per esempio, l'istruzione

```
ANDI.W     #$000F,D1
```

azzera tutti i bit di D1 tranne i 4 bit meno significativi. Questa istruzione potrebbe essere impiegata, ad esempio, per isolare una singola cifra BCD per successivi calcoli matematici. Se l'operando è contenuto nella memoria, esso può essere indirizzato da tutti i modi d'indirizzamento tranne quello relativo al PC.

L'istruzione EOR (*Exclusive OR:* OR esclusivo) ha limitazioni d'indirizzamento un po' diverse, che non si conformano ai requisiti per AND e OR. Essa richiede un registro di dati per la locazione di sorgente. Inoltre, la destinazione dev'essere una locazione alterabile di dati, come avviene per ANDI e ORI. La forma immediata, EORI, ha i medesimi requisiti d'indirizzamento per l'operando di destinazione. Essa è impiegata per eseguire l'OR esclusivo tra un valore immediato ed un operando contenuto in un registro di dati o nella memoria. Chiaramente, la regolarità nell'indirizzamento presente nella maggior parte delle istruzioni dell'MC68020 manca in una certa misura per le istruzioni logiche di questo processore.

L'istruzione NOT effettua il complemento di ciascun bit dell'operando in un registro di dati o nella memoria, mediante un qualsiasi modo d'indirizzamento della memoria, tranne quello relativo al PC. L'operazione svolta da NOT può essere intepretata come negazione logica, se l'operando consiste di *m* variabili logiche, o come complemento a 1, se l'operando è considerato un valore numerico. La descrizione matematica dell'operazione di complemento a 1 è stata fornita nel cap. 3.

Controllo del sistema mediante istruzioni logiche. Le forme immediate ANDI, EORI e ORI possono indicare, come operando di destinazione, il registro di stato o al registro dei codici di condizione.[1] Queste forme sono di solito utilizzate per il controllo del sistema e saranno discusse nel cap. 10. Si rimanda il lettore a tale capitolo per una discussione più completa.

Esempio 8-1

La Tab. 8.3 mostra alcuni esempi di istruzioni logiche che operano su vari operandi. Per semplicità, negli esempi la lunghezza degli operandi è limitata a 8 bit, e sono mostrate soltanto le operazioni di immediato a registro o da registro a registro. Prima dell'esecuzione di ogni istruzione, il byte meno significativo di D1 contiene il valore {1101 0001}, che rappresenta otto variabili logiche. Similmente, (D2)[7:0] = {1101 0101} all'inizio.

[1] Il registro dei codici di condizione (*Condition Code Register:* CCR) è il byte meno significativo del registro di stato (*Status Register:* SR), cioè SR[7:0].

Tab. 8.3 *Esempi di operazioni logiche.*

Istruzione	Operandi	Risultato
ANDI.B #$F0,D1	{1111 0000}	(D1)[7:0] = {1101 0000}
	AND {1101 0001}	
ORI.B #03,D1	{0000 0011}	(D1)[7:0] = {1101 0011}
	OR {1101 0001}	
NOT.B D1	NOT {1101 0001}	(D1)[7:0] = {0010 1110}
EOR.B D1,D2	{1101 0001}	(D2)[7:0] = {0000 0100}
	EOR {1101 0101}	

Nota:
(D1)[7:0] = {1101 0001} e (D2)[7:0] = {1101 0101} prima dell'esecuzione di ciascuna istruzione.

L'istruzione ANDI, com'è usata nell'esempio, serve per "mascherare" (azzerare) i 4 bit meno significativi di D1. L'istruzione ORI fa l'opposto, cioè pone a {1} i bit designati (bit 0 e 1) mentre lascia invariati gli altri bit nella destinazione. Dopo l'esecuzione di ciascuna di queste istruzioni, il bit N dei codici di condizione sarebbe posto a {1} negli esempi mostrati.

L'istruzione NOT inverte gli 8 bit meno significativi di D1. Se il valore originale in D1 è intepretato come il numero decimale –46 nella notazione in complemento a 1 con 8 bit, allora l'inversione produce +46, come previsto.

L'istruzione di OR esclusivo (EOR) fa sì che una variabile logica nel risultato sia posta a {1} quando le due variabili nelle corrispondenti posizioni di bit degli operandi sono differenti. Per esempio, se D1 contiene la prima lettura di 8 bit di stato provenienti da un dispositivo esterno, mentre D2 contiene i dati di una seconda lettura acquisiti successivamente, qualsiasi variazione nella word di stato sarebbe indicata da un risultato diverso da zero (Z = {0}) dopo l'operazione EOR. Se le letture forniscono dati identici, il risultato sarebbe stato composto da tutti zeri, per cui Z = {1}. Si potrebbe usare un'istruzione di salto condizionato, come BEQ o BNE, dopo l'istruzione EOR, al fine di determinare il percorso successivo del programma.

Esempio 8-2

Per illustrare l'impiego di istruzioni logiche con varie modalità d'indirizzamento, la subroutine di esempio della Fig. 8.1 implementa le equazioni per un "decodificatore da due linee a quattro linee", usando le istruzioni dell'MC68020. Uno di quattro possibili valori di uscita è determinato dal valore di due variabili d'ingresso. Queste sono due variabili logiche, designate come A e B nell'esempio. Esse sono contenute in due byte consecutivi nella memoria, con A nel primo, e sono indirizzate da (A1). L'uscita dev'essere costituita da quattro variabili, designate Y0, Y1, Y2 e Y3, memorizzate in quattro byte consecutivi indirizzati da (A2). Tutte le variabili logiche sono giustificate a destra nelle loro locazioni; cioè, la variabile {A} ha la rappresentazione di memoria {0000 000A}. La tavola della verità e le equazioni sono mostrate nella Tab. 8.4, da cui si nota che soltanto un'uscita può essere {0} per ciascuna coppia di ingressi. Se A e B insieme sono interpretate come un numero binario di due cifre, {BA}, allora Yn = {0} quando il valore dell'ingresso è n, con n = 0, 1, 2 o 3. Ogni altra variabile di uscita vale {1}. Quindi viene selezionata la linea di uscita che corrisponde al valore degli ingressi. In questo caso, la linea selezionata è considerata "VERA" o "ON" quando il suo valore è {0}.

La subroutine nella Fig. 8.1 trasferisce dapprima A e B ai byte meno significativi di D0 e D1, rispettivamente. Il complemento di ciascun ingresso viene formato allora nel bit meno significativo di un altro registro. Ogni equazione per Yn è codificata in maniera diretta. Il risultato calcolato viene registrato nella memoria usando lo spostamento appropriato dall'indirizzo di base contenuto in A2.

Tab. 8.4 *Tavola della verità per il decodificatore.*

INGRESSO		**USCITA**			
B	A	Y0	Y1	Y2	Y3
0	0	0	1	1	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0

Note:
Y0 = NOT (NOT A AND NOT B) = A OR B
Y1 = NOT (A AND NOT B) = NOT A OR B
Y2 = NOT (NOT A AND B) = A OR NOT B
Y3 = NOT (A AND B) = NOT A OR NOT B

```
                          1.         TTL     FIGURA 8.1
                          2.         LLEN    100
00010000                  3.         ORG     $10000
                          4.  *
                          5.  *   DECODIFICATORE DA 2 A 4 LINEE
                          6.  *      INPUT:  (A1.L) = INDIRIZZO DI DUE BYTE
                          7.  *                    CONTENENTI LE VARIABILI LOGICHE
                          8.  *              (A2.L) = INDIRIZZO DI 4 BYTE PER I VALORI DECODIF.
                          9.  *      OUTPUT: 4 BYTE DECODIFICATI NELLA LOCAZIONE
                          10. *              PUNTATA DA (A2)
                          11. *
                          12. *      NOTE:   LE VARIABILI LOGICHE SONO MEMORIZZATE
                          13. *              NEI BIT INFERIORI DEL BYTE.
                          14. *
  # 00000000              15. A      EQU     0
  # 00000001              16. B      EQU     1
  # 00000000              17. Y0     EQU     0
  # 00000001              18. Y1     EQU     1
  # 00000002              19. Y2     EQU     2
  # 00000003              20. Y3     EQU     3
                          21. *
00010000  48E7 F800       22. DECODER MOVEM.L D0-D4,-(SP)      ;SALVA I REGISTRI
                          23. *
00010004  1011            24.        MOVE.B  A(A1),D0          ;LEGGE A
00010006  1229 0001       25.        MOVE.B  B(A1),D1          ;LEGGE B
                          26. *
0001000A  1400            27.        MOVE.B  D0,D2
0001000C  4602            28.        NOT.B   D2
0001000E  0202 0001       29.        ANDI.B  #01,D2            ;COMPLEMENTO DI A
                          30. *
00010012  1601            31.        MOVE.B  D1,D3
00010014  4603            32.        NOT.B   D3
00010016  0203 0001       33.        ANDI.B  #01,D3            ;COMPLEMENTO DI B
                          34. *
0001001A  1800            35.        MOVE.B  D0,D4
0001001C  8801            36.        OR.B    D1,D4
0001001E  1484            37.        MOVE.B  D4,Y0(A2)         ;A .OR. B
                          38. *
00010020  1802            39.        MOVE.B  D2,D4
00010022  8801            40.        OR.B    D1,D4
00010024  1544 0001       41.        MOVE.B  D4,Y1(A2)         ;NOT A .OR. B
                          42. *
00010028  1800            43.        MOVE.B  D0,D4
0001002A  8803            44.        OR.B    D3,D4
0001002C  1544 0002       45.        MOVE.B  D4,Y2(A2)         ;A .OR. NOT B
                          46. *
00010030  8602            47.        OR.B    D2,D3
00010032  1543 0003       48.        MOVE.B  D3,Y3(A2)         ;NOT A .OR. NOT B
                          49. *
00010036  4CDF 001F       50.        MOVEM.L (SP)+,D0-D4       ;RIPRISTINA I REGISTRI
0001003A  4E75            51.        RTS
0001003C                  52.        END
```

Fig. 8.1 *Programma per un decodificatore da due a quattro linee.*

ESERCIZI

8.1.1 Si scriva una semplice subroutine per convertire valori ASCII in BCD e viceversa, usando istruzioni logiche. Questa conversione è stata discussa nel cap. 7.

8.1.2 Si migliori il programma del decodificatore impiegato come esempio in questo paragrafo. Si rende più generale il programma, affinché consenta la decodifica da 4 linee a 16 linee.

8.1.3

Si scriva una subroutine per scambiare i contenuti di due word di *m* bit nella memoria usando soltanto l'istruzione EOR e le opportune istruzioni di trasferimento dei dati.

8.2 ISTRUZIONI DI SCORRIMENTO E DI ROTAZIONE

Le istruzioni di scorrimento e di rotazione dell'MC68020 spostano i bit di un operando a destra o a sinistra del numero di posizioni specificato. Sono disponibili tre distinte possibilità, cioè:

(a) Scorrimenti aritmetici

(b) Scorrimenti logici

(c) Rotazioni

Gli scorrimenti aritmetici a sinistra sono in effetti delle moltiplicazioni di un intero binario senza segno per una potenza di 2. Gli scorrimenti aritmetici a destra eseguono invece la divisione per potenze di 2. Naturalmente, il numero fatto scorrere dev'essere entro un intervallo ammesso, altrimenti il risultato sarebbe errato. Gli scorrimenti logici sono impiegati per far scorrere una *m*-tupla di variabili logiche verso destra o verso sinistra. Le istruzioni di rotazione sono tali che i bit fatti scorrere fuori da un'estremità della *m*-tupla riappaiano all'altra estremità in uno scorrimento *ciclico*.

Istruzione	Dim. di operando	Operazione
ASL	8, 16, 32	X/C ← [←] ← 0
ASR	8, 16, 32	↻ [→] → X/C
LSL	8, 16, 32	X/C ← [←] ← 0
LSR	8, 16, 32	0 → [→] → X/C
ROL	8, 16, 32	C ← [←] ↺
ROR	8, 16, 32	↻ [→] → C
ROXL	8, 16, 32	C ← [←] ← X ↺
ROXR	8, 16, 32	↻ X → [→] → C

Fig. 8.2 *Istruzioni di scorrimento e di rotazione. (Per gentile concessione di Motorola, Inc.)*

La Fig. 8.2 illustra le operazioni degli scorrimenti aritmetici (ASL, ASR), degli scorrimenti logici (LSL, LSR), e delle istruzioni di rotazione (ROL, ROR). Le istruzioni di rotazione hanno varianti estese per lo scorrimento di operandi in precisione multipla. Le istruzioni ROXL e ROXR includono il bit X nello scorrimento ciclico. La sintassi del linguaggio di assemblatore per l'istruzione è presentata nella Tab. 8.5.

Sono disponibili tre formati distinti per designare il conteggio di scorrimento e l'operando da far scorrere. Il conteggio può essere tenuto in un registro o specificato come un valore immediato per gli operandi in un registro di dati. È ammesso uno scorrimento di una posizione di un operando di word nella memoria. Questi casi sono mostrati nella Tab. 8.6. Per esempio, l'istruzione

```
LSR.W       D3,D2
```

fa scorrere l'operando di 16 bit nella word meno significativa di D2 verso destra, del numero di posizioni designato in D3. Il numero in D3 è trattato come in modulo 64, cosicché sono possibili scorrimenti da 0 a 63 posizioni. Naturalmente, dopo 16

Tab. 8.5 *Sintassi del linguaggio assembler per le istruzioni di scorrimento e di rotazione.*

Scorrimento aritmetico a sinistra		*Scorrimento aritmetico a destra*	
ASL.<l>	<Dm>,<Dn>	ASR.<l>	<Dm>,<Dn>
ASL.<l>	#<d>,<Dn>	ASR.<l>	#<d>,<Dn>
ASL	<EA>	ASR	<EA>
Scorrimento logico a sinistra		*Scorrimento logico a destra*	
LSL.<l>	<Dm>,<Dn>	LSR.<l>	<Dm>,<Dn>
LSL.<l>	#<d>,<Dn>	LSR.<l>	#<d>,<Dn>
LSL	<EA>	LSR	<EA>
Rotazione a sinistra		*Rotazione a destra*	
ROL.<l>	<Dm>,<Dn>	ROR.<l>	<Dm>,<Dn>
ROL.<l>	#<d>,<Dn>	ROR.<l>	#<d>,<Dn>
ROL	<EA>	ROR	<EA>

Note:
1. <l> denota B, W o L quando <Dn> è la destinazione; <Dm> o #<d> specifica il conteggio di scorrimento.
2. Quando la destinazione è una locazione di memoria, sono ammessi soltanto operandi lunghi una word per <EA>.
3. Soltanto le modalità d'indirizzamento alterabili di memoria sono ammesse per <EA>, esclusi i modi diretto di registro e relativi al PC.
4. ROXL e ROXR hanno la medesima sintassi delle istruzioni di rotazione.
5. I bit N e Z dei codici di condizione sono impostati in base al risultato; V è azzerato tranne che da ASL.

Tab. 8.6 *Formati degli operandi per scorrimenti e rotazioni.*

Formato dell'operando	Conteggio di scorrimento	Destinazione
<Dm>,<Dn>	(Dm); intervallo 0-63	(Dn)[1]
#<d>,<Dn>	#<d>; intervallo 1-8	(Dn)[1]
<EA>	1	(EA)[15:0]

Note:
1. <l> è B, W o L per operandi di registro.
2. [l] indica i bit corrispondenti nell'operazione.
3. <EA> è un indirizzo alterabile di memoria.
4. Un conteggio di scorrimento nullo in Dm ha un significato speciale.

scorrimenti logici verso sinistra o verso destra in un operando di word, il valore rimasto nella word meno significativa sarà composto da bit tutti zero. L'istruzione

```
LSL.W       #5,D3
```

esegue uno scorrimento logico a sinistra di cinque posizioni della word meno significativa di D3. La lunghezza di uno scorrimento immediato può variare da 1 a 8. La terza forma di specificazione di un operando consente lo scorrimento di un operando di word nella memoria di una posizione alla volta. Può essere trattato qualsiasi operando indicato da una modalità d'indirizzamento alterabile di memoria. Per esempio, l'istruzione

```
ASR         (A2)
```

esegue uno scorrimento aritmetico a destra di una singola posizione sull'operando di 16 bit indirizzato da A2.

Un'istruzione di scorrimento che utilizza il conteggio di scorrimento contenuto nel registro di dati effettua uno *scorrimento dinamico*, poiché il conto può essere modificato sotto il controllo del programma. Un conto di scorrimento zero influirà soltanto sui codici di condizione, come mostrato nell'app. D. Se il conto di scorrimento è diverso da zero, gli scorrimenti logici e aritmetici conservato l'ultimo bit, fatto scorrere fuori nei bit C e X. Le istruzioni di rotazione interessano soltanto il bit C. Le operazioni di rotazione con estensione fanno scorrere il valore precedente del bit X in un'estremità dell'operando, salvando all'altra estremità, nei bit C e X, l'ultimo valore che la rotazione aveva portato fuori.

Dopo ogni operazione di scorrimento o rotazione, i codici di condizione N e Z sono impostati in base al risultato, proprio come avviene per le operazioni aritmetiche. Il codice V di condizione di overflow viene posto a {0} dopo ogni operazione,

tranne quella di scorrimento aritmetico a sinistra. L'istruzione ASL moltiplica l'operando per 2^r se esso viene fatto scorrere di *r* bit a sinistra. Se il risultato supera l'intervallo numerico di un operando di *m* bit con segno, il valore {1} di V indica una condizione di fuori-intervallo. Per numeri senza segno, il bit di riporto C posto a {1} indica un overflow dopo un'istruzione ASL.

Esempio 8-3

Nella Tab. 8.7 sono mostrate varie operazioni di scorrimento e di rotazione. I risultati indicati nella tabella sono stati ottenuti supponendo che il byte meno significativo di D1 contenesse il valore binario binario {1110 0101} e che il bit X fosse {0} prima dell'esecuzione di ciascuna istruzione.

Nel caso dell'istruzione ASR, il bit di segno è esteso a destra in ciascuno scorrimento. Il numero originale era –27 nella notazione in complemento a 2. Esso diviene –14 e poi –7 dopo scorrimenti successivi, simulando quindi la divisione intera per 2 ogni volta. Si noti, tuttavia, che –14 non è il risultato atteso della divisione di –27 per 2. L'impiego dell'istruzione DIVS avrebbe fornito un quoziente di –13 con resto –1. Questo troncamento nella direzione sbagliata si presenterà ogni volta che un intero negativo dispari viene fatto scorrere verso destra.

L'istruzione LSL, con un conteggio di scorrimento di 4, fa scorrere i 4 bit meno significativi del byte nei 4 bit superiori. Il bit del codice di condizione N sarebbe posto a {0} per indicare che il bit più significativo è zero. La rotazione di un byte di quattro posizioni a destra con l'istruzione ROR fa scorrere 4 bit e lascia invariato ciascuno di essi. La rotazione del valore originale con X = {0} di una posizione a destra con l'istruzione ROXR causa lo scorrimento di uno {0} nel bit più significativo. Il bit che è stato fatto scorrere fuori dall'estremità destra viene salvato sia in C che in X. Si noti dalla Fig. 8.2 che, se X = {1} prima che ROXR sia eseguita, allora il bit più significativo dell'operando diverrebbe {1}.

Tab. 8.7 *Esempio di operazioni di scorrimento e rotazione.*

Istruzione	Prima (D1)[7:0]	Dopo (D1)[7:0]	C	X
ASR.B #2,D1	{1110 0101}	{1111 1001}	0	0
LSL.B #4,D1	{1110 0101}	{0101 0000}	0	0
ROR.B #4,D1	{1110 0101}	{0101 1110}	0	–
ROXR.B #1,D1	{1110 0101}	{0111 0010}	1	1

Nota: X = {0} prima che ciascuna istruzione venga eseguita.

Esempio 8-4

La subroutine mostrata nella Fig. 8.3 moltiplica l'intero positivo di 32 bit in D4 per una potenza positiva di 2 specificata nel registro di dati D1. Il numero viene passato alla subroutine in D4, mentre il risultato di 64 bit è contenuto nei registri D5 e D4. Se il risultato causa un riporto da D5, il byte meno significativo di D2 viene posto a –1 per indicare l'overflow; altrimenti, l'indicatore di errore in D2 è zero. Il ciclo è un esempio di scorrimento dinamico, poiché la potenza di 2 in D1 viene usata come un contatore che viene decrementato ad ogni scorrimento.

```
                            1.          TTL     FIGURA 8.3
                            2.          LLEN    100
00010000                    3.          ORG     $10000
                            4.  *
                            5.  *   MOLTIPLICAZIONE PER UNA POTENZA DI 2
                            6.  *       INPUT:  (D1.B) = MOLTIPLICATORE POTENZA DI 2
                            7.  *               (D4.L) = INTERO DA MOLTIPLICARE
                            8.  *                        (DEV'ESSERE POSITIVO)
                            9.  *
                            10. *       OUTPUT: (D5/D4)[64:0] = RISULTATO
                            11. *               (D2.B) = 0 : SUCCESSO
                            12. *                       -1 : RIVELATO ERRORE
                            13. *
00010000  2F01              14. POWER2  MOVE.L  D1,-(SP)        ;SALVA IL REGISTRO
00010002  4285              15.         CLR.L   D5              ;RISULTATO ZERO
00010004  143C 00FF         16.         MOVE.B  #-1,D2          ;PREFISSA STATO PER ERRORE
                            17. *
00010008  E38C              18. LOOP    LSL.L   #1,D4           ;SCORRIM. A SIN. (MOLTIPL. PER 2)
0001000A  E395              19.         ROXL.L  #1,D5           ;RIPORTO DI SCORRIM. DA D4 IN D5
0001000C  6500 000A         20.         BCS     FINE            ;SE RIPORTO DA D5,
                            21. *                               ;  ESCE CON ERRORE
00010010  0401 0001         22.         SUBI.B  #1,D1           ;DECREMENTA IL CONTO
00010014  66 F2             23.         BNE     LOOP            ;  FINCHE' (D1) = 0
                            24. *
00010016  4202              25.         CLR.B   D2              ;IMPOSTA STATO PER SUCCESSO
                            26. *
00010018  221F              27. FINE    MOVE.L  (SP)+,D1        ;RIPRISTINA IL REGISTRO
0001001A  4E75              28.         RTS
                            29. *
0001001C                    30.         END
```

Fig. 8.3 *Programma per la moltiplicazione di un numero per 2^N.*

ESERCIZI

8.2.1 Perché le istruzioni dell'MC68020 consentono di far scorrere fino a 63 posizioni quando l'operando più lungo è di soli 32 bit?

8.2.2 Si definiscano verbalmente e mediante un'equazione le condizioni per il verificarsi di un errore quando un intero *con segno* viene fatto scorrere a sinistra o a destra dalle istruzioni ASL o ASR. Si considerino sia interi pari che interi dispari.

8.2.3 Si determini il valore numerico quando il numero binario

{1110 0101}

viene fatto scorrere di tre posizioni a sinistra.

8.2.4 Si dimostri che la regola corretta per raddoppiare un numero in complemento a 2 è uno scorrimento ciclico a sinistra.

8.2.5 Come si può accedere ai due byte superiori di un registro di 32 bit che dovranno essere utilizzati da un'istruzione dell'MC68020 che opera su un operando lungo un byte?

8.2.6 Si scrivano le subroutine per l'impaccamento e il disimpaccamento di operandi di 4 bit in valori di lunghezza di byte. Si supponga che gli operandi siano contenuti nella memoria.

8.2.7 Si modifichi la subroutine mostrata nell'esempio 8.4 per moltiplicare interi con segno per 2^N.

8.2.8 Si scriva una subroutine per moltiplicare un intero con segno per 2^r, dove r può essere positivo o negativo. Se un valore negativo viene troncato da scorrimenti a destra, si corregga il risultato per ottenere una corretta divisione di interi, se necessario.

8.3 ISTRUZIONI DI MANIPOLAZIONE DI BIT E D'IMPOSTAZIONE DEI FLAG

È comodo in molte applicazioni utilizzare una variabile logica per indicare uno di due possibili risultati di un'operazione. Una variabile logica impiegata in questo modo è denominata *flag* (indicatore). Una variabile di flag indica che si è verificata una certa condizione e serve a comunicare l'accaduto ad una routine che ha il compito di esaminare tale variabile. Per esempio, i flag che indicano i risultati di operazioni aritmetiche per l'MC68020 formano complessivamente il registro dei codici di condizione. In questo caso, un'istruzione di salto condizionato può esaminare uno o più codici di condizione per determinare l'azione da intraprendere successivamente. Un'altra applicazione comune di un flag è l'indicazione dello stato di un dispositivo periferico. La conoscenza di tale stato indicherebbe se il dispositivo è "occupato" o pronto ad accettare il trasferimento di dati. Di solito, un singolo bit viene attivato (posto a {1}) dal dispositivo quando esso è pronto, e questa informazione viene usata dal processore per avviare il trasferimento dei dati. In un'applicazione di questo tipo, la variabile logica può essere definita un *flag di evento*, che serve a sincronizzare l'attività del processore con quella del dispositivo esterno. Le istruzioni dell'MC68020 per il test, l'attivazione, l'azzeramento o la modifica di un singolo bit entro un operando sono utili per il trattamento di tali bit di flag, come pure per altre operazioni.

Un altro impiego importante delle variabili logiche è quello di comunicare informazioni tra programmi indipendenti o tra routine di un sistema operativo o di altri sistemi software avanzati. Le condizioni per cui un flag dovrebbe essere attivato o azzerato possono essere molto complicate e richiedere un certo numero di test condizionali. L'MC68020 dispone di un'istruzione di impostazione del valore in base alla condizione (*Set according to Condition:* Scc), che consente di assegnare il valore ad una variabile per indicare un risultato VERO o FALSO in base ai valori dei codici di condizione.

Quando le routine del sistema operativo o vari processori condividono un'area di memoria, si può presentare un problema di sincronizzazione se una routine o un processore può accedere ad un'area o ad una locazione di memoria mentre un'altra routine o un altro processore sta utilizzando la medesima area o locazione. Per esempio, ciò può accadere in un sistema in tempo reale in cui la sequenza di esecuzione del programma è controllata in parte da eventi esterni, segnalati al processore mediante un'interruzione. In un sistema multiprocessore, il problema della sincronizzazione richiede una soluzione hardware parziale se i processori agiscono indipendentemente. L'istruzione di test e impostazione (*Test and Set*) è utile in tali situazioni.

8.3.1 Istruzioni di manipolazione del bit

Le istruzioni di manipolazione del bit dell'MC68020 operano su un singolo bit in un registro di dati o nella memoria. Ogni istruzione svolge un test su uno specifico bit e imposta solamente il codice di condizione Z in base al valore del bit in esame. Come variabile logica, il bit Z indica il *complemento* del bit designato. L'istruzione BTST (*Bit TeST:* test del bit) imposta il bit Z in base allo stato del bit esaminato. Le istruzioni BCHG, BCLR e BSET impostano il bit Z conformemente alle rispettive funzioni, che sono quelle di cambiare, azzerare o porre a {1} il bit designato. Le istruzioni di manipolazione del bit sono mostrate nella Tab. 8.8.

Il numero del bit da esaminare può essere contenuto in un registro di dati per la specificazione dinamica oppure può essere specificato come un valore "dinamico" nell'istruzione. Per esempio, se

(D1) = $0000 0001

e

(D2) = $0000 88FF

per specificare il bit 1 del registro D2, allora l'istruzione

```
BCHG        D1,D2
```

produce dapprima l'assegnazione del valore {0}, il complemento di (D2)[1], al codice di condizione Z. Dopodiché, l'operando in D2 diviene $0000 88FD, poiché il bit 1 di D2 viene complementato dall'operazione. Il valore in D1 dovrebbe essere

Tab. 8.8 *Istruzioni di manipolazione di bit.*

Sintassi	**Operazione**
Cambiamento del bit BCHG <Dn>,<EA> BCHG #<bn>,<EA>	 Z = NOT(bn) THEN bn ← NOT(bn)
Azzeramento del bit BCLR <Dn>,<EA> BCLR #<bn>,<EA>	 Z = NOT(bn) THEN bn ← {0}
Attivazione del bit BSET <Dn>,<EA> BSET #>bn>,<EA>	 Z = NOT(bn) THEN bn ← {1}
Test del bit BTST <Dn>,<EA> BTST #>bn>,<EA>	 Z = NOT(bn)

Note:
1. Se <Dn> è la destinazione, allora la lunghezza è 32 bit; altrimenti, la lunghezza dell'operando di destinazione è di un byte.
2. <bn> è il numero del bit dell'operando.
3. BTST ammette tutte le modalità d'indirizzamento per l'operando di destinazione, tranne quella diretta di registro d'indirizzo (soltanto modi di dati).
4. BCHG, BCLR e BSET ammettono soltanto le modalità d'indirizzamento alterabili di dati per la destinazione.

un numero compreso tra 0 e 31 poiché la destinazione è un altro registro di dati. L'istruzione

```
BCLR        #1,D2
```

avrebbe il medesimo effetto su D2.

Quando l'operando è contenuto nella memoria, è ammessa soltanto la lunghezza di un byte per l'operando. Le istruzioni BCHG, BCLR e BSET ammettono soltanto le modalità d'indirizzamento alterabili di dati. Sono pertanto proibiti il modo diretto di registro d'indirizzo e l'indirizzamento relativo al contatore di programma. Invece, l'istruzione BTST consente tutte le modalità d'indirizzamento, tranne quella diretta di registro d'indirizzo.

Esempio 8-5

La subroutine mostrata nella Fig. 8.4 genera la parità dispari per un singolo carattere ASCII contenuto in una locazione indirizzata da A1 e lascia il risultato nella medesima locazione. Il bit di parità dispari (bit 7) è posto a {0} quando il numero di bit {1} nel carattere è dispari; altrimenti, il bit di parità è posto a {1}. Ne consegue che, in tal modo, il numero di bit non zero nel byte è sempre un numero dispari.

La routine inizia ponendo (D0) = $FF, indicando che il bit di parità dovrebbe essere {1}. Nel ciclo, quando si incontra un bit {0} non si ha alcun effetto e viene esaminato il bit successivo nella sequenza. I bit sono esaminati nell'ordine 6, 5, 4, 3, 2, 1 e 0. Quando viene trovato un bit {1}, (D0) viene complementato. Poiché (D0) era stato inizializzato per indicare che era richiesto un bit di parità, il fatto di trovare 1, 3 o 5 bit di valore {1} farà sì che la routine complementi il valore iniziale in D0. Altrimenti, se 0, 2, 4 o 6 bit hanno il valore {1}, (D0) indicherà che il bit di parità dev'essere posto a {1}. L'istruzione BSET serve a tal fine.

```
                            1.         TTL      FIGURA 8.4
                            2.         LLEN     100
00010000                    3.         ORG      $10000
                            4.  *
                            5.  *   GENERATORE DI PARITA'
                            6.  *      INPUT:   (A1.L) = INDIRIZZO DEL CARATTERE
                            7.  *
                            8.  *      OUTPUT:  IL BIT DI PARITA' (BIT 7) E' SOMMATO AL CARATTERE
                            9.  *               IN (A1)
00010000  48E7 C000        10.  PARITY MOVEM.L  D0/D1,-(SP)       ;SALVA I REGISTRI
00010004  123C 0006        11.         MOVE.B   #6,D1             ;PREDISPONE IL CONTATORE
00010008  103C 00FF        12.         MOVE.B   #$FF,D0           ;INDICATORE DI PARITA' = TUTTI 1
                           13.  *
0001000C  0311             14.  LOOP   BTST     D1,(A1)           ;ESAMINA IL BIT
0001000E  6700 0004        15.         BEQ      NEXT              ;SE DIVERSO DA ZERO
00010012  4600             16.         NOT.B    D0                ;  ALLORA COMPLEMENTA L'INDICATORE
                           17.                                    ;  DI PARITA'
00010014  51C9 FFF6        18.  NEXT   DBRA     D1,LOOP           ;CONTINUA, FINCHE' (D1) = -1
                           19.  *
00010018  4A00             20.         TST.B    D0                ;SE LA PARITA' DEV'ESSERE 0,
0001001A  6700 0006        21.         BEQ      FINE              ;  ALLORA SALTA (NON CAMBIA IL BIT)
0001001E  08D1 0007        22.         BSET     #7,(A1)           ;  ALTRIMENTI PONE A 1 IL BIT DI PARITA'
                           23.  *
00010022  4CDF 0003        24.  FINE   MOVEM.L  (SP)+,D0/D1       ;RIPRISTINA I REGISTRI
00010026  4E75             25.         RTS
                           26.  *
00010028                   27.         END
```

Fig. 8.4 *Subroutine per generare un bit di parità dispari.*

8.3.2 L'istruzione d'impostazione in base alla condizione

L'istruzione Scc assegna il valore {1} a tutti gli 8 bit di una locazione di destinazione se la condizione "cc" è vera. Altrimenti, se la condizione è falsa, tutti gli

8 bit vengono posti a {0}. La condizioni (riporto avvenuto, riporto non avvenuto, ecc.) sono le medesime di quelle dell'istruzione DBcc discussa nel cap. 6. Per esempio, se il bit del codice di condizione Z è {1}, l'istruzione d'impostazione se uguale a zero (*Set if EQual:* SEQ)

```
SEQ        D1
```

assegna il valore $FF al byte meno significativo di D1, indicando una condizione VERA per qualsiasi programma che esamini il flag in D1.

La destinazione <EA> per l'istruzione dev'essere una locazione alterabile di dati. Quindi sono ammesse tutte le modalità d'indirizzamento tranne quella diretta di registro e quella relativa al contatore di programma.

Dopo che tutti i bit dell'operando sono stati posti a {1} o a {0}, la sua condizione potrebbe essere esaminata mediante l'istruzione TST.B, che pone Z = {1} quando il flag ha il valore {0000 0000}. In effetti, tutte le istruzioni logiche che operano su un operando di byte possono essere impiegate per agire sulla variabile logica creata dall'istruzione Scc. Per esempio, se D1 contiene il valore $FF in seguito all'istruzione SEQ appena mostrata, allora l'istruzione

```
EORI.B     #$FF,D1
```

produce la condizione Z = {1} e inverte il flag.

8.3.3 L'istruzione di test e impostazione

L'istruzione TAS (*Test And Set:* esamina e imposta) è usata per esaminare e modificare il valore di un operando di byte contenuto in un registro di dati o nella memoria. La sua operazione è definita nella Fig. 8.5(a); l'istruzione ha la forma simbolica:

```
TAS        <EA>
```

Se l'operando è zero, il bit del codice di condizione Z è posto a {1}; altrimenti, il bit Z viene azzerato. Se il bit 7 dell'operando è {1}, allora N viene posto a {1}. Sotto questo aspetto, TAS si comporta in maniera molto simile all'istruzione TST (Test) operante sul valore di un byte. Tuttavia, dopo che l'operando è stato esaminato dall'istruzione TAS e che N e Z sono stati impostati di conseguenza, il bit più significativo dell'operando viene posto a {1}.

Per esempio, se il byte usato come flag e indirizzato da A1 ha il valore iniziale $00, l'istruzione

```
TAS        (A1)
```

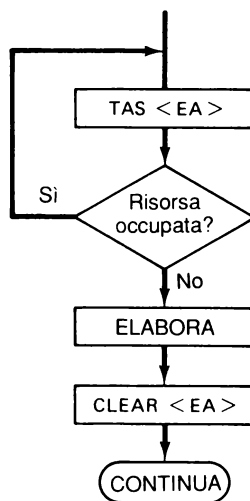
produce la condizione Z = {1} e cambia l'operando in $80 dopo che l'istruzione è stata eseguita. Se il valore iniziale zero indicava che un'area di memoria o qualche

Fig. 8.5
L'istruzione TAS.

```
IF (EA) = 0
        THEN poni Z = {1}
        ELSE poni Z = {0}
IF (EA)[7] = {1}
        THEN poni N = {1}
        ELSE poni N = {0}
Poni (EA)[7] = {1}
```

Note:
(1)(EA) è un operando di lunghezza di byte indirizzato da uno dei modi d'indirizzamento alterabile di dati.
(2)Il ciclo di lettura-modifica-scrittura è indivisibile.

(a) Operazione TAS (TAS <EA>).

TAS <EA>
Sì
Risorsa occupata?
No
ELABORA
CLEAR <EA>
CONTINUA

(b) *Uso tipico.*

altra risorsa era libera per l'uso, il bit del codice di condizione Z indica tale situazione dopo che l'istruzione TAS è stata eseguita. Tuttavia, ora il flag è stato alterato. Un test successivo del flag indicherebbe che la risorsa in uso. Questo test successivo potrebbe essere effettuato da un altro programma che viene eseguito in modo concorrente (magari attivato da un'interruzione) o da un altro processore in un sistema multiprocessore.

Si noti che se una variabile di flag posta a $80 venisse esaminata in un ciclo come il seguente:

```
LOOP  TAS   (A1)    ; esamina il flag
      BNE   LOOP    ; salta indietro se non azzerato
```

allora l'istruzione successiva nella sequenza non potrebbe essere eseguita finché qualche altro programma o processore non avrà azzerato il flag. Qualora il flag fosse già attivato, come in questo esempio, l'istruzione TAS non lo modificherebbe.

Come si rileva dalle note in Fig. 8.5(a), l'operando che viene esaminato da TAS non può essere contenuto in un registro d'indirizzo né può essere riferito mediante l'indirizzamento relativo al contatore di programma. Inoltre, i bit C e V sono sempre azzerati dall'operazione TAS. Il grafo di flusso nella Fig. 8.5(b) illustra un impiego tipico dell'istruzione TAS, seguita da un test condizionale. Qui un programma sta effettuando un test per determinare se qualche risorsa, come una stampante parallela o un'area di memoria, è disponibile per l'utilizzazione. Probabilmente il test condizionale sarà realizzato mediante un'istruzione di salto condizionato, che causerà la ripetizione del ciclo finché il flag non sarà azzerato. Con un impiego siffatto, il processore che esegue il test sarà occupato ad eseguire il ciclo finché non si presenterà un'interruzione che consenta l'azzeramento del flag o finché quest'ultimo non verrà azzerato da un altro processore.

Se la risorsa da condividere è d'importanza critica per il proseguimento dell'esecuzione del programma che effettua il test, è necessario un ciclo di attesa con l'istruzione TAS. Allorché la risorsa diverrà disponibile, l'elaborazione potrà continuare utilizzandola. Il flag dovrebbe essere azzerato al termine dell'elaborazione, al fine di lasciare libera la risorsa per altri utenti. Quindi l'istruzione TAS potrebbe essere usata come un mezzo per impostare i flag impiegati per le comunicazioni tra processi, come le routine in tempo reale, o tra routine d'interruzione del sistema operativo. Comunque, essa ha un altro impiego: quello di *sincronizzare* gli accessi ad una risorsa condivisa quando la temporizzazione è un fattore importante. È proprio la caratteristica di temporizzazione dell'istruzione TAS che svolge un ruolo preminente in molti sistemi multiprocessore, in cui più processori condividono risorse quali le aree di memoria. I sistemi multiprocessore saranno considerati più dettagliatamente nel par. 12.6.

Sincronizzazione mediante l'istruzione TAS. L'istruzione TAS è usata per impedire l'accesso ad una risorsa condivisa da parte di altri programmi quando un determinato programma ha il controllo di tale risorsa. Questa condizione è talvolta denotata come "esclusione" (*lockout*). Quando è possibile l'accesso hardware ad una risorsa condivisa, come nei sistemi in multielaborazione, potrebbe sorgere un problema se due processori esaminassero contemporaneamente il flag ed entrambi considerassero disponibile la risorsa in questione. Al fine di prevenire tale possibilità, l'istruzione TAS ha un ciclo di lettura-modifica-scrittura indivisibile. Una volta che l'operando è stato indirizzato dall'MC68020 che esegue l'istruzione

TAS, il bus di sistema non sarà dispnibile per alcun altro dispositivo, incluso un altro processore, finché l'istruzione TAS non sarà stata completata. Il primo processore che esegue l'istruzione TAS ha il controllo della risorsa hardware finché il flag non viene azzerato. Quindi gli accessi sono sincronizzati al livello hardware. I dettagli del funzionamento dell'hardware dell'MC68020 saranno discussi nel cap. 13.

Esempio 8-6

La subroutine mostrata nella Fig. 8.6 illustra l'impiego dell'istruzione TAS per allocare ed "escludere" un blocco di locazione di memoria per il programma chiamante. La memoria è segmentata in otto blocchi di 256 byte; i blocchi sono numerati come 0, 1, 2, ..., 7. Il primo byte di ciascun blocco contiene un flag di esclusione utilizzato dall'istruzione TAS.

In entrata alla subroutine, l'etichetta FREEMEM deve definire l'inizio dell'area di spazio libero. Fino ad otto blocchi di memoria di 256 byte sono esaminati per determinare quale di essi è disponibile all'uso. Se viene individuato un blocco libero, il suo byte di esclusione viene impostato da TAS, il suo indirizzo è riportato in A1, mentre D1 è posto a {0} per indicare che è stato trovato un blocco libero. Se non è stato individuato alcun blocco libero, D1 riporterà un valore diverso da zero.

```
                           1.          TTL     FIGURA 8.6
                           2.          LLEN    100
00010000                   3.          ORG     $10000
                           4.  *
                           5.  *   CONDIVISIONE DELLE RISORSE CON L'ISTRUZIONE TAS
                           6.  *       INPUT:  FREEMEM = INDIRIZZO DELL'AREA DI SPAZIO LIBERO
                           7.  *
                           8.  *       OUTPUT: (D1.B) = 0     : TROVATO UN BLOCCO
                           9.  *                      NON 0 : NESSUN BLOCCO LIBERO
                          10.  *               (A1.L) = INDIRIZZO DEL BLOCCO
                          11.  *
   = 00020000             12.  FREEMEM EQU     $20000          ;PREDISPONE L'AREA LIBERA
00010000  2F03            13.  SHARE   MOVE.L  D3,-(SP)        ;SALVA IL REGISTRO
00010002  4201            14.          CLR.B   D1              ;IMPOSTA STATO PER "TROVATO"
00010004  227C 00020000   15.          MOVE.L  #FREEMEM,A1     ;INDIRIZZO INIZIALE DI MEMORIA
0001000A  7607            16.          MOVEQ   #7,D3           ;PREDISPONE IL CONTATORE
                          17.  *
0001000C  4AD1            18.  LOOP    TAS     (A1)            ;SE IL BLOCCO E' LIBERO,
0001000E  6700 000E       19.          BEQ     FOUND           ; ALLORA ESCE CON INDIRIZZO
00010012  D3FC 00000100   20.          ADDA.L  #256,A1         ; ALTRIMENTI AVANZA AL BLOCCO SUCC.
                          21.  *                               ; ED ESAMINA IL FLAG
00010018  51CB FFF2       22.          DBRA    D3,LOOP         ;CONTINUA FINCHE' (D3) = -1
                          23.  *
0001001C  1203            24.          MOVE.B  D3,D1           ;IMPOSTA STATO PER "NON TROVATO"
                          25.  *
0001001E  261F            26.  FOUND   MOVE.L  (SP)+,D3        ;RIPRISTINA IL REGISTRO
00010020  4E75            27.          RTS
                          28.  *
00010022                  29.          END
```

Fig. 8.6 *Subroutine di allocazione di memoria.*

ESERCIZI

8.3.1

Si specifichi il test di bit necessario per determinare quanto segue:

(a) Un intero è pari.
(b) Un intero con segno, lungo *m* bit, è negativo.
(c) Un carattere è una lettera alfabetica maiuscola o minuscola in codice ASCII.

8.3.2

Si confrontino le seguenti istruzioni con quelle equivalenti di manipolazione del bit:

```
(a) ANDI.W    #7FFF,D2
(b) ORI.W     #$8000,D2
(c) EORI.W    #$8000,D2
(d) TST.W     D2
```

8.3.3

Si specifichino i passi necessari per generare i valori aritmetici (o logici) {1} e {0} dalle condizioni VERO e FALSO impostate dalle istruzioni Scc.

8.3.4

Si scriva una subroutine per modificare una stringa di caratteri alfabetici (ASCII) da maiuscolo a minuscolo, o viceversa. Si consulti l'app. A, che contiene l'insieme di caratteri.

8.3.5

Si confrontino le operazioni hardware e software dell'istruzione

```
TAS       <EA>
```

con quelle dell'istruzione

```
BSET      #7,<EA>
```

Si determinino le condizioni in cui un ciclo di attesa per "occupato", realizzato mediante le istruzioni TST e BNE seguite da un'istruzione BSET anziché da TAS, potrebbe non essere sufficiente per fornire l'esclusione di una risorsa condivisa. Si considerino sia sistemi con un singolo processore che multiprocessore.

8.3.6

Si consideri una "mappa di bit" contenente *N* bit, che indicano la disponibilità di *N* blocchi di 256 byte nella memoria. Se un bit è {0}, allora il blocco corrispondente è libero. Si supponga che tale mappa di bit sia contenuta nella memoria in una locazione fissa e che sia noto il primo indirizzo dell'area di memoria contigua. Si scriva una subroutine che individui la prima sequenza di *k* blocchi contigui, riporti l'indirizzo iniziale di tali blocchi e ponga a {1} i bit appropriati nella mappa di bit per indicare i blocchi di memoria utilizzati. Se vengono trovati meno di *k* blocchi, o se *k* non è un intero compreso tra 1 e *N*, allora, la subroutine dovrà riportare un'indicazione di errore al programma chiamante.

8.4 ISTRUZIONI DI CAMPO DI BIT

Le operazioni logiche presentate all'inizio di questo capitolo operano su bit contenuti in operandi di byte, word o longword. I singoli bit sono indirizzati dal numero di bit entro l'operando. Quando un gruppo di bit di lunghezza arbitraria entro un operando dev'essere indirizzato per eseguire test od operazioni logiche o aritmetiche, la raccolta di bit dev'essere isolata. Un metodo per ottenere tale isolamento consiste nel far sì che un programma trasferisca in un registro di dati l'operando che contiene il gruppo di bit e successivamente faccia scorrere fuori gli altri bit non interessati dall'operazione. Tuttavia, l'MC68020 dispone di un insieme di istruzioni che risultano più comode per isolare un gruppo di bit. Queste istruzioni operano direttamente su un gruppo di bit contigui denominato *campo di bit*. Le istruzioni di campo di bit trattano in effetti tale gruppo come un operando indirizzabile.

La Fig. 8.7 definisce l'indirizzamento di un bit e di un campo di bit nella memoria. Un singolo bit viene indirizzato specificando l'indirizzo di base del suo byte nella memoria ed il conteggio del numero di bit a partire dal bit meno significativo nel byte. Per contro, i valori del *campo di bit* sono specificati da tre parametri nelle istruzioni che operano su di essi.

Gli specificatori sono i seguenti:

(a) L'*indirizzo di base* di un byte nella memoria, che è il primo byte di un array di bit contenente il campo di bit.

(b) La posizione o *offset* relativo all'indirizzo di base dei bit più significativo nel campo (bit [0]).

(c) La *larghezza* del campo, che specifica il numero di bit contigui nel campo.

Si noti che in Fig. 8.7(a) i bit nel campo di larghezza w sono numerati da 0 a w − 1, a partire dal bit più significativo, denominato *bit di base* del campo. La numerazione è opposta a quella per i bit in operandi di byte, word o longword.

La Fig. 8.7(b) definisce i termini che sono applicati ai campi di bit ed agli array di bit. Il *campo di bit*, altresì noto come stringa di bit, è composto da 1 a 32 bit. Un *array di bit* è una struttura di dati che contiene un certo numero di campi di bit. I campi entro l'array possono essere di lunghezza fissa o variabile, a seconda dell'applicazione. L'*indirizzo di base* specifica il bit [0] dell'array. Il campo di bit stesso inizia dalla locazione di bit nella memoria all'indirizzo

(indirizzo di base) + <offset>

che specifica il bit [0] del campo di bit.

Dati di bit

7 0 | 7 0 | 7 0 | 7 0

Byte n − 1 | 7 6 5 4 3 2 1 0 | Byte n + 1 | Byte n + 2

Indirizzo di base

Numero di bit

Dati del campo di bit

Bit di base

7 0 | 7 0 | 7 0 | 7 0

Byte n − 1 | Byte n | | 0 1 2 3 .... w-1 |

Offset — Offset — Larghezza

−3 −2 −1 | 0 1 2 ...

Indirizzo di base

(a)

Campo di bit: una stringa composta da 1 a 32 bit consecutivi entro un array.

Array di bit: una raccolta di bit che possono essere indirizzati in incrementi di bit.

Indirizzo di base: l'indirizzo del byte che contiene il bit [0] dell'array.

Numero del campo di bit: l'offset relativo dal bit più significativo dell'indirizzo di base.

Offset del campo di bit: il numero del campo di bit del bit più significativo del campo. L'intervallo è da -2^{31} a $2^{31} - 1$ nella memoria, se l'offset è contenuto in un registro di dati. Come valore immediato, l'intervallo è 0-31.

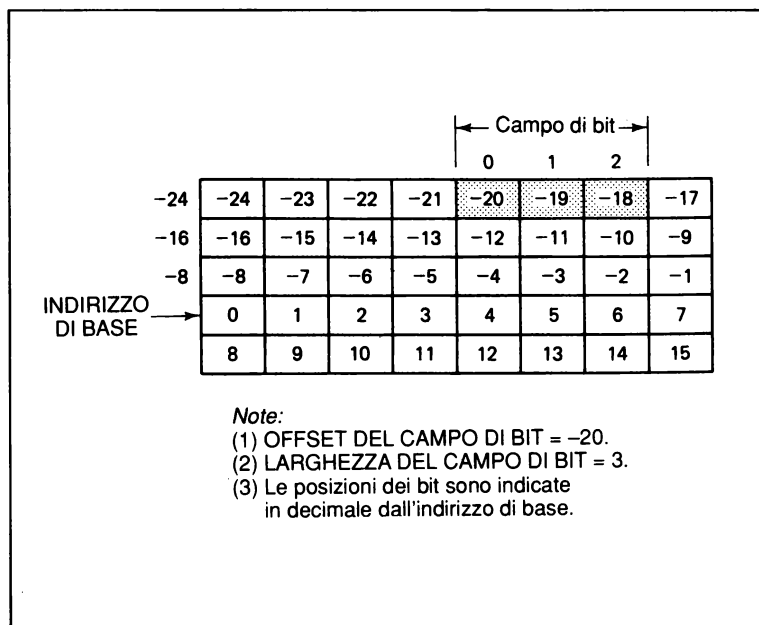
Larghezza del campo di bit: il numero di bit nel campo di bit da 1 a 32.

Fig. 8.7 *(a) Dati di bit e dati di campo di bit nella memoria; (b) definizioni di campo di bit.*

Fig. 8.8
Esempio dell'organizzazione di un campo di bit nella memoria.

Campo di bit: 0 1 2

−24	−24	−23	−22	−21	−20	−19	−18	−17
−16	−16	−15	−14	−13	−12	−11	−10	−9
−8	−8	−7	−6	−5	−4	−3	−2	−1
INDIRIZZO DI BASE →	0	1	2	3	4	5	6	7
	8	9	10	11	12	13	14	15

Note:
(1) OFFSET DEL CAMPO DI BIT = −20.
(2) LARGHEZZA DEL CAMPO DI BIT = 3.
(3) Le posizioni dei bit sono indicate in decimale dall'indirizzo di base.

Nella Fig. 8.7(a), l'offset può essere positivo o negativo. Come esempio, la Fig. 8.8 mostra un array di bit che si estende da 24 bit più in basso nella memoria fino a 15 bit più in alto dall'indirizzo di base. Il campo (evidenziato dall'ombreggiatura) consiste di 3 bit, con i parametri:

<offset> = −20

<larghezza> = 3

Nella Fig. 8.8 il *numero del campo di bit* del bit 1 è:

<offset> + 1 = −19

come mostrato. Questo è l'offset dal bit di base dell'array di bit. Il campo di bit occupa i bit a partire da:

(indirizzo di base) + <offset>

fino a:

(indirizzo di base) + <offset> + <larghezza − 1>

cioè i numeri di campo di bit da −20 a −18 in questo esempio.

8.4.1 Operazioni di campo di bit

Le istruzioni di campo di bit dell'MC68020 sono definite nella Tab. 8.9. Queste istruzioni possono essere suddivise in tre gruppi, come segue:

(a) Istruzioni che modificano, azzerano, pongono a {1} o esaminano un campo di bit in un array di bit.

(b) Istruzioni che estraggono o inseriscono un campo di bit contenuto in un array di bit.

(c) BFFFO, che individua il primo (1) in un campo di bit entro un array di bit.

Tab. 8.9 *Istruzioni di campo di bit: (a) Sintassi delle istruzioni.*

SINTASSI	OPERAZIONE
Test di campo di bit e modifica BFCHG <EA>{offset:w}	Complementa il campo di bit di larghezza w.
Test del campo di bit e azzeramento BFCLR <EA>{offset:w}	Azzera il campo di bit di larghezza w, cioè pone a {0} ciascun bit.
Estrazione del campo di bit con segno BFEXTS <EA>{offset:w},<Dn>	Estrae il campo di bit e lo trasferisce in <Dn>; giustificato a destra ed esteso di segno a 32 bit.
Estrazione del campo di bit senza segno BFEXTU <EA>{offset:w},<Dn>	Estrae il campo di bit e lo trasferisce in <Dn>; giustificato a destra ed esteso di zero a 32 bit.
Ricerca del primo {1} nel campo di bit BFFFO <EA>{offset:w},<Dn>	Pone in <Dn> la posizione del primo bit trovato uguale a {1}.
Inserimento del campo di bit BFINS <Dn>,<EA>{offset:w}	Trasferisce il campo di bit dai bit meno significativi di <Dn> nel campo di bit designato in <EA>.
Impostazione a {1} del campo di bit BFSET <EA>{offset:w}	Assegna il valore {1} ad ogni bit del campo di bit di larghezza w.
Test del campo di bit BFTST <EA>{offset:w}	Esamina il campo di bit e modifica i bit del codice di condizione in base ai valori trovati.

Nota: La posizione di bit per l'istruzione BFFFO è:
<Dn> = <offset> + (offset al primo bit = {1})
se viene trovato un {1} nel campo. Altrimenti, se nessun {1} viene trovato nel campo, la posizione suddetta è:
<Dn> = <offset> + <w>

Tab. 8.2 *Istruzioni di campo di bit: (b) Specificazione delle istruzioni di campo di bit.*

Indirizzamento <EA>	
	Modalità d'indirizzamento
BFCHG, BFCLR, BFINS, BFSET	Modo diretti di registro di dati e alterabili di controllo; tutti tranne An, (An)+, –(An) e relativo al PC.
BFEXTS, BFEXTU, BFFFO, BFTST	Modi diretti di registro di dati e modi di controllo; tutti tranne An, (An)+ e –(An).
Offset e larghezza	
Offset	(a) 0-31 come valore immediato; (b) da -2^{31} a $2^{31} - 1$ in un registro di dati
Larghezza w	1-32 come valore immediato o in un registro di dati

Tab. 8.2 *Istruzioni di campo di bit: (c) Impostazione dei codici di condizione.*

Istruzioni	Bit di codice di condizione
BFXXX (tutte)	V = C = {0} X = invariato
BFCHG, BFCLR, BFSET	N = {1} se il bit più significativo del campo era {1} *prima* che il campo fosse cambiato, azzerato o posto a {1}; N = {0} altrimenti. Z = {1} se il bit più significativo del campo era {1} *prima* che il campo fosse cambiato, azzerato o posto a {1}; Z = {0} altrimenti.
BFEXTS, BFEXTU, BFFFO, BFTST	N = {1} se il bit più significativo del campo è {1}; N = {0} altrimenti. Z = {1} se tutti i bit del campo sono zero; Z = {0} altrimenti.
BFINS	N e Z sono modificati in base al valore del campo *inserito*.

La forma generale della prima categoria è:

BFXXX <EA>{offset:w}

per BFCHG, BFCLR, BFSET e BFTST, in cui w è la larghezza (*width*) del campo di bit. L'indirizzo effettivo <EA>, che specifica l'indirizzo di base dell'array di bit, può essere designato come un registro di dati o una locazione di memoria. La Tab. 8.9(b) mostra che un array di bit nella memoria può essere definito mediante qualsiasi modalità d'indirizzamento tranne quella diretta di registro d'indirizzo, (An)+ e –(An) per l'istruzione BFTST (*Bit Field TeST:* test del campo di bit). Le altre istruzioni (BFCHG, BFCLR e BFSET) non possono impiegare questi modi di registro d'indirizzo né l'indirizzamento relativo al PC. Questa limitazione sull'impiego dell'indirizzamento relativo al PC non giunge inattesa, poiché l'operando del campo di bit è una locazione di destinazione, soggetta ad essere modificata da queste istruzioni. BFTST può usare l'indirizzamento relativo al PC poiché il campo di bit non viene alterato da questa istruzione; soltanto i codici di condizioni vengono modificati allorché essa viene eseguita. L'istruzione determina se tutti i bit del campo sono {0}, una condizione indicata da Z = {1}. Inoltre, essa assegna a N il valore del bit [0] del campo di bit. Come mostrato nella Tab. 8.9(c), i codici di condizione N e Z sono modificati in base al valore del campo di bit prima che BFCHG, BFCLR o BFSET modifichi il campo di bit.

Il campo di bit è definito dall'offset e dalla larghezza w nell'istruzione. La larghezza può essere definita come un valore immediato o in un registro di dati come un valore da 1 a 32 bit. Anche un offset può essere specificato, sia come valore immediato che come valore in un registro di bit. Come operando immediato, l'offset dev'essere un numero positivo nell'intervallo da 0 a 31. L'offset in un registro di dati è considerato un intero in complemento a 2 di 32 bit, nell'intervallo da -2^{31} a $2^{31} - 1$. Quindi l'istruzione

```
BFTST     (A6){D1:4}
```

specifica l'indirizzo di base in A6 mediante l'indirizzamento indiretto di registro d'indirizzo. L'offset è contenuto in D1 per un campo di bit con una larghezza di 4 bit.

Le istruzioni BFEXTS, BFEXTU e BFINS. Un campo di bit può essere estratto da un array di bit mediante le istruzioni BFEXTS e BFEXTU. Si usa BFEXTS (*Bit Field EXTtract Signed:* estrai campo di bit con segno) quando il campo di bit rappresenta un intero con segno. L'istruzione

```
BFEXTS    (A0){0:14},D1
```

prende 14 bit dell'array di bit definito dall'indirizzo di base in A0 e trasferisce in D1 il campo di bit, giustificandolo a destra. Il bit [0] del campo di bit (cioè, il bit [13] in D1) viene esteso (copiato) in (D1)[31:14]. Per contro, l'istruzione BFEXTU (*Bit Field EXTtract Unsigned:* estrai campo di bit senza segno)

```
BFEXTU    (A0){0:14},D1
```

produce il seguente risultato:

(D1)[13:0] = campo di bit

(D1)[31:14] = {0}

indipendentemente da quale fosse il bit più significativo nel campo di bit prima dell'operazione. I codici di condizione N e Z vengono modificati in base al valore del campo di bit prima dell'estrazione.

L'istruzione BFINS (*Bit Field INSert:* inserisci campo di bit) copia un campo di bit da un registro di dati in un altro registro di dati o nella memoria. Essa svolge quindi l'operazione opposta a quella di BFEXTU se il campo di bit è l'unico contenuto del registro di dati di destinazione. L'istruzione di esempio

```
BFINS      D2,(A2){14:23}
```

trasferisce il campo di bit (D2)[22:0] nel campo di bit che inizia dall'indirizzo di *bit* (A2) + 14 fino a

(A2) + 14 + (23 − 1) = (A2) + 36

Il codice di condizione Z è posto a {1} se il campo di bit D2 contiene tutti zeri. Se il bit più significativo del campo di bit in D2 (cioè, (D2)[22]) è {1}, allora il codice di condizione N viene posto a {1}. Altrimenti, se nessuno dei due casi si presenta, Z = {0} e N = {0}.

L'istruzione BFFFO. L'istruzione BFFFO (*Bit Field Find First One:* trova il primo {1} del campo di bit) è usata per trovare l'eventuale numero del campo di bit della posizione del bit più significativo nel campo di bit che contiene un {1}. L'istruzione di esempio

```
BFFFO      (A2){6:11},D1
```

ricerca il primo {1} tra i bit in un campo largo 11 bit. Se un {1} viene trovato, il numero del campo di bit viene posto in D1. Se il bit [n] del campo di bit è {1}, il risultato in D1 è:

(D1) = 6 + n

dove 6 è l'offset dal bit 0 dell'array di bit; n = 0, 1, 2, ..., 10. Se nessun bit del campo di bit è {1}, il valore del registro di destinazione sarà sostituito col valore:

(D1) = <offset> + w

dove w è la larghezza del campo di bit. Nell'esempio, se ciascun bit è {0} nel campo, allora si ha:

(D1) = 6 + 11 = 17

che è l'offset del bit che segue il campo di bit nell'array di bit che inizia dall'indirizzo (A2). Ora il valore in D1 può essere usato come un offset per ricercare il successivo campo di bit nell'array di bit. Se campi di bit consecutivi di lunghezza w = 11 sono zero, l'istruzione

```
BFFFO    (A2){D1:11},D1
```

potrebbe essere usata per indicizzare attraverso i campi di bit entro l'array di bit che inizia dall'indirizzo (A2). Mentre l'istruzione viene eseguita in un ciclo, (D1) cambia come segue:

(D1): 17, 28, 39, ...

finché non viene trovato un {1} in uno dei campi di bit. Quando un {1} viene trovato, il codice di condizione Z sarà posto a {0}. Il codice di condizione N viene posto a {1} se il bit più significativo (bit [0]) del campo di bit era {1}.

Esempio 8-7

La Fig. 8.9 mostra i risultati di varie istruzioni di campo di bit. Il contenuto di D1 è il valore binario:

$1001\ 1100\ 0000\ 1101\ 1101\ 0010\ 0000\ 0001_2$

equivalente a 9C0D D201 in esadecimale, prima dell'esecuzione delle istruzioni BFCHG, BFCLR, BFEXTS, BFEXTU, BFINS, BFSET e BFTST. Il campo di bit è largo 12 bit e inizia in (D1)[26]. Allora il campo di bit è:

$1000\ 0001\ 1011_2$

cioè 81B in esadecimale. I risultati in D1 dopo l'esecuzione di ciascuna istruzione sono mostrati nei commenti per le istruzioni BFCHG, BFCLR e BFSET. Soltanto BFTST imposta i codici di condizione N e Z.

Le istruzioni BFEXTS e BFEXTU estraggono il campo di bit da (D1) e lasciano il risultato in D3. BFEXTS estende di segno il campo di bit ($81B) a 32 bit. In questo caso, come con le altre istruzioni descritte finora, N = {1} e Z = {0} in base al campo di bit in D1 prima che le istruzioni siano eseguite. Il campo di bit:

$0001\ 1000\ 1111_2$

cioè $18F, viene inserito nell'array di bit in D1 dall'istruzione BFINS mostrata nella Fig. 8.9. Il registro di dati D3 contiene il campo di bit da inserire prima che venga eseguita l'istruzione BFINS. Per questa istruzione, N = {0} e Z = {0} in base al risultato del campo di bit ($18F) che viene inserito.

```
                                 1.          TTL      FIGURA 8.9
00010000                         2.          ORG      $10000
                                 3.  *
    * 00000063                   4.  RETURN  EQU      $0063
                                 5.  *
                                 6.  *  ARRAY DI BIT  10011100000011011101001000000001
                                 7.  *  CAMPO DI BIT        ************
                                 8.  *
                                 9.  *  ARRAY DI BIT $9C0DD201 (HEX)
                                10.  *  CAMPO DI BIT $81B (12 BIT)
                                11.  *  RISULTATI IN ESADECIMALE
                                12.  *
00010000  223C 9C0DD201         13.          MOVE.L   #$9C0DD201,D1     ;CREA L'ARRAY DI BIT
00010006  2401                  14.          MOVE.L   D1,D2             ;SALVA L'ORIGINALE
                                15.  *
00010008  EAC1 014C             16.          BFCHG    D1[5:12]          ;(D1)=$9BF25201
                                17.  *                                  ; N=1, Z=0
0001000C  2202                  18.          MOVE.L   D2,D1
0001000E  ECC1 014C             19.          BFCLR    D1[5:12]          ;(D1)=$98005201
                                20.  *                                  ; N=1, Z=0
00010012  2202                  21.          MOVE.L   D2,D1
00010014  EEC1 014C             22.          BFSET    D1[5:12]          ;(D1)=$9FFFD201
                                23.  *                                  ; N=1, Z=0
00010018  2202                  24.          MOVE.L   D2,D1
0001001A  E8C1 014C             25.          BFTST    D1[5:12]          ;N=1, Z=0
                                26.  *
0001001E  2202                  27.          MOVE.L   D2,D1
00010020  EBC1 314C             28.          BFEXTS   D1[5:12],D3       ;(D3)=$FFFFF81B
                                29.  *                                  ; N=1, Z=0
00010024  2202                  30.          MOVE.L   D2,D1
00010026  E9C1 314C             31.          BFEXTU   D1[5:12],D3       ;(D3)=$0000081B
                                32.  *                                  ; N=1, Z=0
0001002A  2202                  33.          MOVE.L   D2,D1
0001002C  EDC1 014C             34.          BFFFO    D1[5:12],D3       ;(D3)=$00000005
                                35.  *                                  ; N=1, Z=0
                                36.  *
                                37.  *       INSERIMENTO DI CAMPO (D3)[11:0]=$18F
                                38.  *
00010030  2202                  39.          MOVE.L   D2,D1             ;ARRAY ORIGINALE
00010032  263C 0000018F         40.          MOVE.L   #$18F,D3          ;CAMPO DA INSERIRE
00010038  EFC1 314C             41.          BFINS    D3,D1[5:12]       ;(D1)=$98C72D01
                                42.                                     ; N=0, Z=0
                                43.  *
0001003C  4E4F                  44.          TRAP     #15               ;RITORNA AL MONITOR
0001003E  0063                  45.          DC.W     RETURN
                                46.  *
00010040                        47.          END
```

Fig. 8.9 *Esempi di istruzioni di campo di bit.*

8.4.2 Applicazioni delle istruzioni di campo di bit

Le istruzioni di campo di bit si rivelano maggiormente utili quando si ha a che fare con un variabile di lunghezza arbitraria (1-32 bit). Questa variabile, rappresentata come un campo di bit, non dev'essere necessariamente allineata su un confine di byte, di word o di longword nella memoria. In alcune applicazioni, il campo di bit può essere considerato come composto di singoli bit; in altri casi, la variabile rappresentata dal campo di bit è considerata come una stringa di bit raggruppati.

Quando vari bit del registri di stato dell'MC68020 o vari bit del registro di stato per un dispositivo periferico sono considerati come un campo di bit, le istruzioni BFCHG, BFCLR, BFSET e BFTST possono essere utilizzate da una routine del

sistema operativo per manipolare i bit in gruppo. Per esempio, i bit d'interruzione del registro di stato (*Status Register:* SR) dell'MC68020, definito nel cap. 4, sono trattati come un numero binario di 3 bit che rappresenta il livello d'interruzione attivo. Se il contenuto di SR viene trasferito ad un registro di dati, questi bit d'interruzione, che costituiscono la cosiddetta "maschera d'interruzione", possono essere azzerati (livello 0) o posti a 1 (livello 7) dalle appropriate istruzioni di campo di bit. Dopo la manipolazione, il contenuto del registro di dati sostituirebbe il valore precedente nel registro di stato, per variare il livello d'interruzione della CPU.

Le istruzioni di estrazione del campo di bit e di inserimento del campo di bit sono utilizzate per costruire nuove stringhe di bit o variabili intere a partire da quelle esistenti. L'istruzione BFEXTS, per esempio, potrebbe essere usata per creare una variabile intera con segno di 32 bit dal valore di 12 bit letto da un convertitore analogico/digitale (A/D) e registrato in una locazione di memoria.[2] Dopo che il valore è stato esteso di segno, si può utilizzare qualsiasi istruzione aritmetica del processore MC68020 per manipolare la variabile. Nelle applicazioni di comunicazione, si può usare BFINS per inserire un campo di bit di lunghezza arbitraria in un array di bit che rappresenta un messaggio. Il messaggio da trasmettere tra i computer potrebbe consistere di un array di bit con vari campi di bit di lunghezza differente. Senza le istruzioni di campo di bit, potrebbe essere necessaria una considerevole quantità di mascheramento e di scorrimento dei campi di bit per creare l'array di bit del messaggio.

L'istruzione BFINS trova impiego anche nella grafica a rappresentazione di bit (*bit mapping*). In quest'applicazione, lo schermo è rappresentato nella memoria come una matrice di bit, cioè un array bidimensionale di bit. Ciascun bit corrisponde sullo schermo ad un punto, denominato *pixel* (contrazione di *picture element:* elemento dell'immagine). Per tracciare un'immagine sullo schermo, i bit appropriati nella matrice di bit devono essere posti a {1}. Questi bit rappresentano i punti illuminati sullo schermo quando la matrice di bit viene trasmessa ad un terminale grafico. L'istruzione d'inserimento del campo di bit BFINS è usata per attivare i bit appropriati, indipendentemente dall'organizzazione della memoria in byte, word o longword; infatti l'istruzione agisce soltanto sui bit interessati.

Il concetto di *mappa di bit* o di *matrice di bit* è utile per l'allocazione di spazio nella memoria o in un'unità a disco. Per esempio, il sistema operativo CP/M (*Control Program for Microcomputers:* programma di controllo per microcomputer) impiega la mappa di bit di allocazione per indicare i settori del disco che sono disponibili durante l'esecuzione del programma.[3] Per questo e per altri sistemi operativi, un {1} nella mappa di bit indica che una certa area sul disco non è disponibile, magari perché un programma vi ha già memorizzato dei dati. Se un valore nella mappa di bit è {0}, allora la corrispondente area del disco è libera per l'uso.

---

[2] Alcuni dispositivi, come i convertitori A/D, possono avere dati di uscita che rappresentano valori interi ma la cui lunghezza non è multipla di 8 bit. Convertitori A/D diversi possono avere uscite di 12, di 14 o di 20 bit.

[3] CP/M è un marchio registrato della Digital Research, Inc.

L'istruzione BFFFO è comoda per distinguere le aree libere del disco da quelle utilizzate. L'esempio 8.8 illustra l'applicazione dell'istruzione BFFFO per tale scopo.

Sono possibili molti altri impieghi delle istruzioni di campo di bit. Ad esempio, certi linguaggi ad alto livello consentono di definire variabili di lunghezza arbitraria. Quando un programma che utilizza variabili di lunghezza arbitraria viene compilato, il compilatore può generare istruzioni di campo di bit per prelevare, manipolare o memorizzare le variabili. Tali applicazioni e molte altre sono discusse in vari riferimenti bibliografici relativi a questo capitolo, riportati nell'app. E.

Esempio 8-8

La subroutine di Fig. 8.10 trova il primo bit {0} in una mappa di bit di *M* byte il cui indirizzo iniziale è fornito in A0 prima dell'esecuzione del programma. Il numero di byte è fornito in D0 e dev'essere un multiplo di quattro byte. Nel metodo impiegato in questo esempio, la mappa di bit viene dapprima invertita per consentire l'uso dell'istruzione BFFFO (trova il primo {1} nel campo di bit). Dopo tale inversione, il ciclo all'etichetta FFONE ricerca un {1} nei campi di bit larghi 32 bit. D2 contiene la larghezza, che è fissata a 32 bit. Il valore in D1 è l'offset dall'indirizzo in A0, e questo valore varia in incrementi di 32 mentre il ciclo viene eseguito. Questo incremento si ottiene sostituendo il valore in D1 col valore (D1) + (D2) se nessun bit {1} viene trovato in un campo di bit, poiché, dopo l'esecuzione dell'istruzione BFFFO, si ha (D1) = <offset> + <larghezza> se nessun {1} è stato trovato. Altrimenti, se l'istruzione BFFFO trova un {1} in qualsiasi campo largo 32 bit, D1 conterrà il valore dell'offset.

Dopo l'esecuzione della ricerca, D3 contiene l'offset di bit dell'eventuale primo bit che era {1} nella mappa di bit invertita. Se non viene trovato alcun bit di valore {1}, allora (D3) = –1. La mappa di bit già invertita viene sottoposta nuovamente ad inversione per ripristinarne i valori originali, prima che la subroutine restituisca il controllo al programma chiamante. Quindi il valore in D3 rappresenta l'offset del bit che era {0} nella mappa di bit originale. Di solito, ciò indicherebbe un'area libera della memoria o un'area disponibile sul disco se i bit zero nella mappa di bit corrispondevano ad aree allocabili ad un programma da parte del sistema operativo.

Un altro metodo potrebbe eliminare l'inversione della mappa di bit originale. Per esempio, si potrebbe usare l'istruzione BFTST per trovare il primo valore zero o per trovare un numero arbitrario di bit consecutivi {0}. Il programmatore può scegliere tra vari metodi di ricerca nella mappa di bit, a seconda delle informazioni che devono essere ottenute. Vari problemi presentati negli esercizi di questo paragrafo potrebbero essere risolti in modi diversi, in base alle preferenze del programmatore.

```
                         1.         TTL     FIGURA 8.10
00010000                 2.         ORG     $10000
                         3.  *
                         4.  *   TROVA IL PRIMO (0) IN UNA MAPPA DI BIT
                         5.  *   LUNGA (D0)*8 BIT
                         6.  *
                         7.  *      INPUT:  (A0.L) = INDIRIZZO DELLA MAPPA DI BIT
                         8.  *              (D0.L) = NUMERO DI BYTE NELLA MAPPA DI BIT
                         9.  *                       (RICHIESTI MULTIPLI DI 4)
                        10.  *
                        11.  *      OUTPUT: (D3.L) = OFFSET DA (A0) AL PRIMO BIT (0),
                        12.  *                       SE TROVATO
                        13.  *                     = -1 SE NESSUN BIT (0) TROVATO
                        14.  *
                        15.  *      NOTA:   LA MAPPA DI BIT VIENE INVERTITA IN ENTRATA
                        16.  *              E REINVERTITA PRIMA DI USCIRE
                        17.  *
00010000 48E7 E840      18.         MOVEM.L D0-D2/D4/A1,-(SP)      ;SALVA I REGISTRI
00010004 2248           19.         MOVEA.L A0,A1                  ;INDIRIZZO INIZIALE
00010006 2800           20.         MOVE.L  D0,D4                  ;SALVA LA LUNGHEZZA
                        21.  *
                        22.  *   INVERTE LA MAPPA DI BIT
                        23.  *
00010008 4619           24.  INVERT1 NOT.B  (A1)+                  ;INVERTE (D0) BYTE
0001000A 5380           25.         SUBQ.L  #1,D0
0001000C 66 FA          26.         BNE     INVERT1
0001000E 2248           27.         MOVEA.L A0,A1                  ;RIPRISTINA L'INDIRIZZO INIZIALE
                        28.  *
                        29.  *   TROVA IL PRIMO (1) (IL PRIMO (0) NELLA MAPPA DI BIT ORIGINALE)
                        30.  *
00010010 2004           31.         MOVE.L  D4,D0                  ;RIPRISTINA LA LUNGHEZZA
                        32.
                        33.
00010012 E448           34.         LSR     #2,D0                  ;NUMERO DI CAMPI
                        35.  *                                     ;  LUNGHI 32 BIT
00010014 7200           36.         MOVE.L  #0,D1                  ;OFFSET
00010016 7420           37.         MOVE.L  #32,D2                 ;LARGHEZZA
00010018 EDD0 1862      38.  FFONE  BFFFO   (A0){D1:D2},D1         ;(D1) = OFFSET AL PRIMO
                        39.  *                                     ;  BIT (1), SE ESISTE
0001001C 6600 0010      40.         BNE     FOUND                  ;  TROVATO SE Z=(0)
00010020 5380           41.         SUBQ.L  #1,D0
00010022 6700 0004      42.         BEQ     NOBITSZ                ;  SALTA SE NESSUNO TROVATO
00010026 60 F0          43.         BRA     FFONE                  ;    ALTRIMENTI ESAMINA CAMPO SUCC.
                        44.
                        45.  *
00010028 76FF           46.  NOBITSZ MOVE.L #-1,D3                 ;NESSUNO TROVATO
0001002A 6000 0004      47.         BRA     INVERT2
0001002E 2601           48.  FOUND  MOVE.L  D1,D3                  ;OFFSET AL BIT (0)
                        49.                                        ;  NELLA MAPPA DI BIT ORIGINALE
00010030 4619           50.  INVERT2 NOT.B  (A1)+
00010032 5384           51.         SUBQ.L  #1,D4
00010034 66 FA          52.         BNE     INVERT2
                        53.  *
00010036 4CDF 0217      54.         MOVEM.L (SP)+,D0-D2/D4/A1
0001003A 4E75           55.         RTS
0001003C                56.         END
```

Fig. 8.10 *Ricerca di mappa di bit con l'impiego di BFFFO.*

ESERCIZI

8.4.1

Si confrontino le operazioni delle seguenti istruzioni:

(a) BFCHG, BCHG, NOT
(b) BFCLR, BCLR, CLR, ANDI #MASK,D0
(c) BFSET, BSET, Scc, ORI #MASK,D0
(d) BFTST, BTST, TST, TAS
(e) BFEXTS, EXT

MASK contiene un'opportuna configurazione di {1} e {0} per azzerare o porre a {1} i bit come richiesto dall'applicazione. Si confrontino l'operazione, la lunghezza dell'operando, e l'impostazione dei codici di condizione per ciascun gruppo di istruzioni.

8.4.2 Si supponga che la memoria contenga i seguenti valori esadecimali:

Offset di bit (decimale)	Contenuto (esadecimale)
–24	59
–16	C1
–8	67
0	9C
+8	0D
+16	D2
+24	01
+32	E6

L'indirizzo di base è \$20002. Si scriva la configurazione di bit e s'illustrino i risultati delle seguenti istruzioni quando:

(D0) = \$FFFFFFF2
(D1) = \$00000017
(D2) = \$12345678
(A2) = \$00020002

prima dell'esecuzione di ciascuna istruzione.

```
(a) BFTST    (A2){D1:15}
(b) BFCLR    $20002{5:12}
(c) BFSET    (A2){30:5}
(d) BFEXTU   $20002{4:14},D3
(e) BFEXTS   $20002{4:14},D3
(f) BFINS    D2,(A2){D0:D1}
(g) BFFFO    (A2){6:11},D1
```

8.4.3 Si scriva una routine per creare i caratteri ASCII che corrispondono alle cifre esadecimali in un valore di 32 bit contenuto in D0. Procedendo da sinistra a destra, si memorizzi la stringa ASCII in un array di bit puntato da A1.

8.4.4 Si scriva una routine per moltiplicare due array di valori di 24 bit, elemento per elemento. I prodotti devono essere numeri con o senza segno di 64 bit. Come ingresso, un valore di flag indica il tipo di aritmetica.

8.4.5 Usando la struttura di mappa di bit dell'esempio 8.8, si scrivano delle routine che svolgano le seguenti azioni:

(a) Determinare l'offset di bit del primo bit di una stringa con N valori {0} consecutivi. N è un valore tra 1 e 32. Se nessuna stringa viene trovata, ritornare con un valore di flag di errore.

(b) Contare il numero di {1} e il numero di {0} nella mappa di bit e riportare i valori al programma chiamante.

8.4.6

Si faccia riferimento agli esercizi del par. 8.3. Si scriva una routine per risolvere il problema 8.3.6 usando le appropriate istruzioni di campo di bit.

CAPITOLO 9

TECNICHE DI PROGRAMMAZIONE

Nei capitoli precedenti, l'insieme di istruzioni dell'MC68020 è stato introdotto suddividendo le istruzioni in categorie. In tali categorie sono state discusse istruzioni impiegate nella creazione di programmi per trasferire dati, eseguire calcoli, o fornire semplici funzioni. Perlopiù questi programmi erano progettati per soddisfare i requisiti di una particolare applicazione, come la conversione da ASCII a binario. In questo capitolo saranno invece esplorate varie tecniche di programmazione che si rivelano utili nella creazione di programmi più sofisticati. Un accento particolare sarà posto sulla potenza della capacità d'indirizzamento dell'MC68020.

Alcune istruzioni dell'MC68020, come DIVU e MULU, non ammettono operazioni dirette sui registri d'indirizzo. Questa limitazione non è affatto severa, poiché i contenuti dei registri d'indirizzo e dei registri di dati possono essere scambiati facilmente. In aggiunta, l'MC68020 dispone comunque di istruzioni speciali per il trattamento degli indirizzi. Queste istruzioni saranno presentate nel primo paragrafo poiché forniscono la flessibilità necessaria per le tecniche di programmazione avanzate.

Una tecnica speciale richiede la creazione di *codice indipendente dalla posizione*. L'indirizzamento relativo al contatore di programma e l'indirizzamento di registro di base possono essere impiegati nei programmi, affinché la loro esecuzione sia indipendente dall'indirizzo iniziale nella memoria. Questi argomenti saranno discussi nel par. 9.2.

Il trattamento di tipiche *strutture di dati* quali array e liste richiede l'impiego di metodi avanzati di programmazione. Anche se soltanto alcuni dei molti argomenti che riguardano la creazione e la manipolazione di strutture di dati saranno discussi nel par. 9.3, la potenza e la flessibilità delle istruzioni dell'MC68020 saranno comunque evidenziate.

Le *subroutine* sono state introdotte nel cap. 6 come un metodo comune per assistere il programmatore nella creazione di programmi modulari. Il collegamento tra un programma chiamante ed una subroutine, tramite un indirizzo di ritorno posto sullo stack di sistema, è già stato discusso con sufficiente dettaglio in tale capitolo. Il passaggio di valori di dati o di indirizzi tra i moduli di programma era ottenuto servendosi dei registri del processore per contenere i valori. Nel par. 9.4 saranno discussi alcuni metodi più sofisticati per trasferire valori di dati. Saranno presentate anche le tecniche per creare un frame di stack mediante le istruzioni LINK e UNLK e per scrivere subroutine rientranti.

9.1 ISTRUZIONI PER IL TRATTAMENTO DI INDIRIZZI

Tranne che per valori contenuti in registri o per valori specificati dal modo d'indirizzamento immediato, il riferimento agli operandi delle istruzioni dell'MC68020 viene effettuato mediante i loro indirizzi. Inoltre si fa una distinzione tra dati e indirizzi, poiché hanno linee distinte per i segnali di uscita e sono contenuti in registri di dati e registri d'indirizzo, rispettivamente. Le linee dei segnali d'indirizzo specificano una locazione nella memoria (o nello spazio di I/O del sistema), mentre le linee dei segnali di dati sono usate per trasferire valori. Internamente, sia valori di dati che indirizzi possono essere "manipolati" con varie istruzioni del processore.

L'insieme di istruzioni dell'MC68020 dispone di istruzioni che operano specificamente su indirizzi. Esse comprendono istruzioni per confrontare due indirizzi (CMPA), per eseguire operazioni aritmetiche (ADDA, SUBA) e per trasferire indirizzi (MOVEA, LEA, PEA). A parte CMPA, esse non modificano il registro dei codici di condizione.

Per tutte le operazioni che riguardano indirizzi, il campo d'indirizzamento valido per l'MC68020 si estende da 0 a $FFFFFFFF (esadecimale). Per istruzioni che ammettono operandi di lunghezza di word, gli indirizzi di 16 bit vengono estesi di segno a 32 bit prima di essere utilizzati dall'istruzione. Pertanto, gli indirizzi corti (16 bit) hanno un intervallo di validità da 0 a $7FFF oppure da $FFFF8000 a $FFFFFFFF.

9.1.1 Trattamento di indirizzi aritmetici

Le istruzioni ADDA (*ADD Address:* somma indirizzo), SUBA (*SUB Address:* sottrai indirizzo) e CMPA (*CMP Address:* confronta indirizzo) agiscono su un operando di sorgente che può essere indirizzato mediante qualsiasi modalità. Comunque, l'operando di destinazione dev'essere contenuto in un registro d'indirizzo. La sintassi e le operazioni di queste istruzioni sono mostrate nella Tab. 9.1. Esse sono simili alle istruzioni ADD, SUB e CMP, che operano su valori di dati.

L'istruzione ADD somma un valore contenuto in un registro di dati o nella memoria al valore nel registro d'indirizzo di destinazione. Un impiego comune del-

Tab. 9.1 *Operazioni di ADD, SUBA e CMPA.*

Istruzione	**Sintassi**	**Operazione**	**Codici di condizione interessati**
Somma indirizzi	ADDA.<l> <EA>,<An>	(An) ← (An) + (EA)	Nessuno
Sottrai indirizzi	SUBA.<l> <EA>,<An>	(An) ← (An) – (EA)	Nessuno
Confronta indirizzi	CMPA.<l> <EA>,<An>	(An) – (EA)	N, Z, V, C

Note:
1. <l> denota W o L soltanto.
2. Se è specificato un operando di word (W), esso viene esteso di segno a 32 bit.
3. Tutte le modalità d'indirizzamento sono ammesse per <EA>.

l'istruzione ADDA è quello di sommare una costante, che è specificata dal modo d'indirizzamento immediato, al valore contenuto nel registro d'indirizzo interessato. Per esempio, l'istruzione

```
ADDA.L    #20,A1
```

incrementa (A1) di 20 quando viene eseguita. In un ciclo, questa istruzione fa sì che A1 indirizzi un elemento ogni venti in una struttura di dati. Comunque, le istruzioni che operano su indirizzi forniscono una flessibilità molto maggiore, poiché l'operando di sorgente può essere specificato mediante qualsiasi modalità d'indirizzamento. Per esempio, un'istruzione come la seguente:

```
ADDA.L    A2,A2
```

raddoppia il valore in A2. Essa potrebbe essere impiegata per convertire un numero d'ingresso contenuto in A2 in un indice che opera su una tabella di word di 16 bit. Per esempio, se (A2) = $100 indica la 256-ma word in un blocco di memoria, allora 2 x $100, cioè $200, è l'indirizzo di tale word, espresso come offset (in byte) dalla locazione iniziale del blocco.

L'istruzione SUBA forma la differenza tra un registro d'indirizzo di destinazione e l'operando di sorgente. L'istruzione

```
SUBA.L    D1,A1
```

lascia in A1 il valore di 32 bit di

(A1) – (D1)

Come l'istruzione ADDA, anche SUBA non modifica i codici di condizione.

L'istruzione CMPA serve a determinare l'ordine di due indirizzi (inferiore, uguale o superiore). I codici di condizione sono impostati come per l'istruzione CMP, discussa nel cap. 6. Gli indirizzi dovrebbero essere considerati come interi senza segno e i test sui bit C e Z hanno la medesima interpretazione già fornita nel cap. 6. Anche le istruzioni di salto condizionato BHI (*Branch if HIgher:* salta se superiore), BLS (*Branch if Lower or Same:* salta se inferiore o identico), BNE (*Branch if Not Equal:* salta se diverso), o BEQ (*Branch if EQual:* salta se uguale) sono utili per effettuare i test di confronto di indirizzi. Per esempio, le istruzioni

```
CMPA.L    A1,A2    ; forma (A2) – (A1)
BHI       LOOP     ; salta se (A2) > (A1)
```

producono un salto a LOOP se (A2) è maggiore di (A1); altrimenti, l'esecuzione procederà con l'istruzione successiva nella sequenza.

L'istruzione CMPA effettua il calcolo di

(destinazione) – (sorgente)

e imposta i codici di condizione in base al risultato. La Tab. 9.2 riassume le istruzioni di salto condizionato applicabili ad un'istruzione CMPA.

Anche le varianti dell'istruzione ADDQ (*ADD Quick:* somma rapidamente) e SUBQ (*SUBtract Quick:* sottrai rapidamente) possono essere usate rispettivamente per sommare o sottrarre dal contenuto di un registro d'indirizzo una costante nell'intervallo da 1 a 8. L'operando di sorgente è il valore immediato della costante, come discusso nel cap. 7. Queste istruzioni sono del tipo ad una sola word; esse sono impiegate per eseguire in modo efficiente l'incremento o il decremento di un indirizzo.

Tab. 9.2 *Confronto di indirizzi.*

Istruzione	**Salto**	**Condizione**
CMPA.L A1,A2	BHI (superiore)	$(A2) > (A1)$
	BLS (inferiore o identico)	$(A2) \leq (A1)$
	BCC (superiore o identico)	$(A2) \geq (A1)$
	BCS (basso)	$(A2) < (A1)$
	BNE (diverso)	$(A2) \neq (A1)$
	BEQ (uguale)	$(A2) = (A1)$

Esempio 9-1

La subroutine di Fig. 9.1 conta il numero di interi negativi di 8 bit in un blocco o tabella di byte consecutivi nella memoria. L'indirizzo iniziale delo blocco è passato alla subroutine in A0, mentre A1 deve contenere l'indirizzo dell'ultimo byte + 1 all'inserimento. Ogni byte viene successivamente esaminato ed il conteggio è accumulato nella word meno significativa di D0.

In ciascun ciclo, A0 viene incrementato di 1 con l'istruzione ADDA. Il test termina quando il valore nel registro d'indirizzo A0 diventa uguale all'indirizzo finale in A1.

```
                          1.          TTL     FIGURA 9.1
                          2.          LLEN    100
00010000                  3.          ORG     $10000
                          4.  *
                          5.  *   DETERMINA IL NUMERO DI BYTE NEGATIVI IN UN BLOCCO
                          6.  *
                          7.  *       INPUT:  (A0.L) = INDIRIZZO INIZIALE DEL BLOCCO
                          8.  *               (A1.L) = INDIRIZZO FINALE DEL BLOCCO + 1
                          9.  *
                          10. *       OUTPUT: (D0.W) = NUMERO DI BYTE NEGATIVI
                          11. *
00010000  2F08            12. CNTNEG  MOVE.L  A0,-(SP)        ;SALVA IL REGISTRO
00010002  4240            13.         CLR.W   D0              ;AZZERA IL CONTATORE
                          14. *
00010004  4A10            15. LOOP    TST.B   (A0)            ;SE IL BYTE E' NON NEGATIVO,
00010006  6A00 0004       16.         BPL     NEXT            ;  ALLORA ELABORA IL BYTE SUCC.
0001000A  5240            17.         ADDQ.W  #1,D0           ;  ALTRIMENTI LO CONTA
0001000C  D1FC 00000001   18. NEXT    ADDA.L  #1,A0           ;INCREMENTA AL BYTE SUCCEESIVO
00010012  B3C8            19.         CMPA.L  A0,A1           ;SE NON HA TERMINATO,
00010014  62 EE           20.         BHI     LOOP            ;  ALLORA CONTINUA
00010016  205F            21.         MOVE.L  (SP)+,A0        ;RIPRISTINA IL REGISTRO
00010018  4E75            22.         RTS
0001001A                  23.         END
```

Fig. 9.1 *Esempi di manipolazione di indirizzo.*

9.1.2 Trasferimento di indirizzi

Le istruzioni MOVEA, LEA, PEA sono impiegate per trasferire indirizzi. L'istruzione MOVEA (*MOVE Address:* trasferisce indirizzo) carica un registro d'indirizzo con un operando che può essere contenuto in un registro o in una locazione di memoria o che può essere specificato come un valore immediato. L'istruzione LEA (*Load Effective Address:* carica indirizzo effettivo) calcola l'indirizzo effettivo di una locazione di memoria e trasferisce il valore dell'indirizzo ad un registro d'indirizzo. L'istruzione PEA (*Push Effective Address:* inserisci indirizzo effettivo) calcola un indirizzo effettivo e lo inserisce in cima allo stack di sistema. LEA e PEA differiscono dalla maggior parte delle istruzioni in quanto viene trasferito l'indirizzo effettivo calcolato per un operando, anziché l'operando stesso. La sintassi e le modalità

Tab. 9.3 *Istruzioni per il trasferimento di indirizzi.*

Istruzione	Sintassi	Lunghezza dell'operando (byte)	Modalità d'indirizzamento	
			Sorgente	Destinazione
Trasferisci indirizzo	MOVEA.<l> <EA>,<An>	16 o 32	Tutte	An
Carica indirizzo effettivo	LEA <EA>,<An>	32	Modi di controllo	An
Inserisci indirizzo	PEA <EA>	32	Modi di controllo	–(SP)

Note:
1. I codici di condizione non sono alterati.
2. <l> denota W o L soltanto.
3. Per operandi lunghi una word, l'operando di sorgente viene esteso di segno a 32 bit, e tutti i 32 bit vengono caricati nel registro d'indirizzo.

d'indirizzamento per queste tre istruzioni che trasferiscono indirizzi sono mostrate nella Tab. 9.3.

L'istruzione di trasferimento dell'indirizzo. L'istruzione MOVEA ha la forma simbolica:

MOVEA <l> <EA>,<An>

dove <l> = W o L, mentre <EA> può essere specificato mediante qualsiasi modalità d'indirizzamento. L'operazione effettua il trasferimento:

$(An)[31{:}0] \leftarrow (EA)$

in cui qualsiasi riferimento di 16 bit viene esteso di segno a 32 bit prima del trasferimento. Quando la sorgente è un registro di dati, un registro d'indirizzo o una locazione di memoria, è il relativo contenuto che viene trasferito. L'istruzione

```
MOVEA.L        TABEL,A1
```

trasferisce in A1 la word di 32 bit contenuta nella locazione etichettata come TABEL. Il valore in questa locazione è considerato un indirizzo; l'etichetta TABEL in questione potrebbe denotare il primo indirizzo di una tabella di indirizzi. Il registro A1 potrà quindi essere usato per far riferimento al valore puntato dall'indirizzo alla locazione TABEL. Per esempio, se l'istruzione MOVEA fornita sopra è seguita dall'istruzione

```
MOVE.W         (A1),D1
```

allora la word di 16 bit riferita dall'operando (indirizzo) alla locazione TABEL viene trasferita nella word meno significativa di D1. Le due operazioni effettuano quindi il trasferimento:

(D1)[15:0] ← ((TABEL)[31:0])[15:0]

Per contro, quando s'impiega la modalità immediata con un'etichetta, viene trasferito l'indirizzo. Nell'istruzione

```
MOVEA.L        #TABEL,A1
```

l'indirizzo alla locazione TABEL, che è l'operando di sorgente, viene trasferito in A1. Quindi la forma immediata carica l'indirizzo della tabella stessa. Senza il simbolo immediato, l'istruzione MOVEA sarebbe usata per caricare in A1 il primo indirizzo contenuto in una tabella di indirizzi che inizia dalla locazione TABEL.

Caricamento dell'indirizzo effettivo. L'istruzione LEA è usata per calcolare un indirizzo effettivo basandosi sulla modalità d'indirizzamento di sorgente e per trasferirlo ad un registro d'indirizzo. La forma simbolica è:

```
LEA        <EA>,An
```

dove <EA> è specificato da una modalità d'indirizzamento di controllo. Quindi non sono ammessi i modi d'indirizzamento diretto, con postincremento e con predecremento. L'istruzione

```
LEA        TABEL,A1
```

produce l'operazione:

(A1)[31:0] ← TABEL

dove TABEL è un indirizzo in un programma in linguaggio assembler. Questa istruzione è equivalente a MOVEA con la forma immediata dell'indirizzo dell'operando di sorgente.

Quando sono impiegate altre modalità d'indirizzamento per l'operando di sorgente nell'istruzione LEA, questa svolge delle funzioni che non sono possibili con altre istruzioni di trasferimento di dati. LEA consente di calcolare un indirizzo durante l'esecuzione del programma e di trasferirlo ad un registro d'indirizzo.

Si consideri la sequenza di istruzioni:

```
LEA       (2,A1,D1.W),A0   ; CALCOLA L'INDIRIZZO
MOVE.W    (A0),D2          ; PONE IL VALORE IN D2
MULU      #4,D2            ; MOLTIPLICA IL VALORE PER 4
MOVE.W    D2,(A0)          ; LO SALVA
```

Tale sequenza calcola dapprima un indirizzo in base alla modalità d'indirizzamento indiretto con indice, cioè:

(A1) + (D1)[15:0] + 2

e lo trasferisce in A0. L'operando di word indirizzato da (A0) viene trasferito in D2, modificato e salvato. Il riferimento indiretto (A0) è più efficiente nell'impiego della memoria di quanto non lo sia il riferimento indiretto con indice (2,A1,D1.W) che richiede una word di estensione per lo spostamento alla sua istruzione. Inoltre, per calcolare l'indirizzo indiretto è richiesto un numero di cicli di macchina minore di quello necessario per calcolare l'indirizzo indretto con indice. In assenza dell'istruzione LEA, entrambe le istruzioni di trasferimento richiederebbero l'indirizzamento indicizzato per calcolare la locazione dell'operando.

Inserimento dell'indirizzo effettivo sullo stack. L'istruzione PEA calcola un indirizzo effettivo ed impiega lo stack di sistema come destinazione. La forma simbolica:

PEA <EA>

effettua il calcolo di un indirizzo effettivo di 32 bit per un operando specificato da una delle modalità d'indirizzamento di controllo. Il valore calcolato viene inserito sullo stack dalla CPU mediante la sequenza:

(SP) ← (SP) – 4

((SP)) ← <EA>

in cui il puntatore dello stack di sistema viene dapprima decremento di 4 e poi usato per puntare alla locazione di longword per <EA>.

Esempio 9-2

La Tab. 9.4 mostra i risultati dell'esecuzione delle istruzioni PEA, LEA e MOVEA. Per l'istruzione PEA, il puntatore di stack viene inizializzato a $0000 7FFE. L'istruzione inserisce (A0) sullo stack come mostrato. Poiché s'impiega lo stack di sistema, i byte più significativi di A0 sono contenuti ad indirizzi inferiori nella memoria. LEA carica A0 con l'operando di sorgente stesso. Per conseguire tale risultato, si dovrebbe utilizzare MOVEA con un valore immediato. Nel prossimo esempio, se l'operando di sorgente fosse specificato per MOVEA come un indirizzo assoluto anziché immediato, il contenuto della locazione di word in $8000 sarebbe trasferito in A0. Così com'è, il valore immediato $8000 viene esteso di segno a $FFFF 8000 prima di essere usato.

Tab. 9.4 *Esempi di manipolazione di indirizzo.*

Contenuto della memoria (esadecimale)	Istruzione	Risultati
4850	PEA (A0)	(\$7FFA) = \$00 (\$7FFB) = \$00 (\$7FFC) = \$10 (\$7FFD) = \$20
41F9 0001 2345	LEA \$00012345,A0	(A0) = \$0001 2345
307C 8000	MOVEA.W #\$8000,A0	(A0) = \$FFFF 8000

Nota: Inizialmente, (A0) = \$0000 1020 e (SP) = \$0000 7FFE.

Esempio 9-3

La Fig. 9.2 (a pagina seguente) elenca le sequenze di istruzioni equivalenti a LEA e PEA. In ogni caso, A0 contiene il primo indirizzo di una tabella che rappresenta blocchi di operandi di lunghezza di byte nella memoria. La word meno significativa di D0 è un indice per selezionare l'indirizzo iniziale di un blocco particolare, quando viene sommato ad (A0). L'OFFSET seleziona l'indirizzo di un particolare byte, quando viene usato per calcolare l'indirizzo effettivo.

ESERCIZI

9.1.1

Sia (A1) = \$0000 1000 prima dell'esecuzione di ciascuna istruzione elencata di seguito. Si calcoli l'indirizzo in A1 dopo che ciascuna istruzione è stata eseguita.

(a) ADDA.W #\$2000,A1
(b) ADDA.L #\$9000,A1
(c) SUBA.W #\$2000,A1

```
                              1.         TTL     FIGURA 9.2
                              2.         LLEN    100
00010000                      3.         ORG     $10000
                              4.  *
  = 0000000A                  5.  OFFSET EQU     10
                              6.  *
                              7.  *   CONFRONTO DI LEA E PEA
                              8.  *
                              9.  *      INPUT:  (A0.L) = INDIRIZZO DELLA TABELLA
                             10.  *              (D0.W) = INDICE NELLA TABELLA
                             11.  *              OFFSET = BYTE ENTRO L'ELEMENTO DI TAB. PREDEFINITA
                             12.  *
                             13.  *      OUTPUT: (A1.L) = INDIRIZZO EFFETTIVO (LEA)
                             14.  *
                             15.  *   OPERAZIONE DI LEA PER CARICARE L'INDIRIZZO DI UN ELEMENTO
                             16.  *
00010000  43F0 000A          17.         LEA     (OFFSET,A0,D0.W),A1
                             18.  *
                             19.  *   CARICA L'INDIRIZZO DI UN ELEMENTO SENZA USARE LEA
                             20.  *
00010004  2248               21.         MOVE.L  A0,A1
00010006  D2C0               22.         ADDA.W  D0,A1
00010008  D2FC 000A          23.         ADDA.W  #OFFSET,A1
                             24.  *
                             25.  *   OPERAZIONE DI PEA PER SALVARE UN INDIRIZZO
                             26.  *
0001000C  4870 000A          27.         PEA     (OFFSET,A0,D0.W)
                             28.  *
                             29.  *   SALVATAGGIO DI UN INDIRIZZO SENZA L'IMPIEGO DI PEA
                             30.  *
00010010  2248               31.         MOVE.L  A0,A1
00010012  D2C0               32.         ADDA.W  D0,A1
00010014  D2FC 000A          33.         ADDA.W  #OFFSET,A1
00010018  2F09               34.         MOVE.L  A1,-(SP)
                             35.  *
0001001A                     36.         END
```

Fig. 9.2 *Confronto delle istruzioni LEA e PEA. (Esempio 9-3)*

9.1.2

I valori esadecimali in A0 e A1 siano i seguenti:

(A0) = $0000 1000
(A1) = $0001 1F00

prima dell'operazione:

CMPA.W   A0,A1

Quale delle istruzioni di salto condizionato, eseguita dopo il confronto, causerebbe un salto?

9.1.3

Se l'istruzione

LEA   (1,A1,D1.W),A2

viene eseguita con

(D1) = $0000 8000
(A1) = $0000 1FFF

qual è l'indirizzo caricato in A2?

9.1.4 Si consideri il segmento di programma:

```
TABLE    EQU   $20100
         MOVE.L      #$20200,TABLE
         MOVE.L      #$11111111,([TABLE])
```

che viene eseguita prima dell'esecuzione di ognuna delle seguenti istruzioni :

```
(a) MOVEA.L   ([TABLE]),A1
(b) LEA       ([TABLE]),A1
(a) MOVEA.L   #TABLE,A1
```

Qual è il risultato di A1 per ciascun caso?

9.1.5 Come si possono usare LEA e PEA per il debugging di un programma verificando gli indirizzi da cui sono prelevati i dati?

9.1.6 Si generalizzi il concetto d'indirizzamento indipendente dalla memoria, per consentire N livelli d'indirizzamento indiretto. S'illustri la sequenza di istruzioni per indirizzare un operando usando tre livelli d'indirizzamento indiretto di memria.

9.2 CODICE INDIPENDENTE DALLA POSIZIONE E INDIRIZZAMENTO DI BASE

Nella maggior parte dei precedenti esempi di programmazione, il programma occupava locazioni fisse nella memoria e l'indirizzo iniziale era definito dalla direttiva di origine (ORG) dell'assemblatore. Se questi programmi dovessero essere spostati in un'altra area di memoria, sarebbe necessario un riassemblaggio con una nuova origine. Il fatto che i programmi non possano essere trasferiti o "rilocati" nella memoria senza riassemblaggio è indesiderabile in alcuni casi, anzi è assolutamente inaccettabile per programmi basati su ROM. In questi casi, i programmi devono poter essere rilocati dopo essere stati assemblati.[1] Alcuni sistemi operativi potrebbero avere l'esigenze di rilocare un programma applicativo anche dopo l'inizio della sua esecuzione.

Un programma è definito *staticamente indipendente dalla posizione* se esso può essere caricato ed eseguito da qualsiasi indirizzo iniziale nella memoria. La maggior parte dei programmi in ROM sono staticamente indipendenti dalla posizione, poiché l'indirizzo iniziale del programma nella ROM è definito dal progettista di sistema in base ai requisiti del sistema. Per esempio, una routine in virgola mobile nella ROM può avere un indirizzo iniziale $2000 in un certo sistema, ma iniziare dalla locazione $10000 in un altro, magari perché il secondo sistema richiede un'area molto maggiore di RAM contigua. La scrittura di *codice indipendente dalla posizione* è una tecnica di codifica di una routine in modo da renderne arbitrario l'indirizzo iniziale. Una delle principali peculiarità di un programma di questo tipo è

[1] La rilocazione effettuata mediante un editor di collegamento è discussa in vari riferimenti biliografici relativi a questo capitolo, elencati nell'app. E.

che esso non contiene alcun indirizzo assoluto tranne quelli imposti dalle definizioni dell'hardware, come gli indirizzi dei dispositivi di I/O.

L'*indipendenza dinamica dalla posizione* consente di trasferire un programma anche dopo che la sua esecuzione è stata avviata. Ciò è richiesto nei sistemi con *memoria virtuale*. In parole povere, un sistema con memoria virtuale permette di scrivere programmi senza tener conto delle limitazioni della memoria fisica. Se è richiesta una rilocazione, il sistema operativo e la circuiteria di gestione della memoria controlleranno automaticamente l'indirizzamento effettivo (fisico).

9.2.1 Codice indipendente dalla posizione con (PC)

Quando una locazione riferita in un programma si trova ad una distanza fissa dall'istruzione in cui compare il riferimento, si può impiegare la modalità d'indirizzamento relativo al contatore di programma per creare un codice indipendente dalla posizione. A patto che lo spostamento relativo non venga modificato, i programmi saranno eseguiti correttamente ovunque nella memoria. Se tutti i riferimenti alla memoria impiegano l'indirizzamento relativo al PC, il programma sarà *dinamicamente* indipendente dalla posizione, poiché gli indirizzi effettivi saranno calcolati durante l'esecuzione di ciascuna istruzione. Il trasferimento di un programma ed il suo riavviamento dal punto esatto in cui era stato temporaneamente sospeso non causeranno alcun problema, se il programma e i dati saranno trasferiti insieme come un blocco unico. Purtroppo è difficile scrivere dei programmi per l'MC68020 che impieghino soltanto il modo d'indirizzamento relativo al PC, poiché la maggior parte delle istruzioni non possono fare riferimento in questo modo agli operandi di destinazione nella memoria.

La Tab. 9.5 mostra i tipi di istruzioni e i riferimenti di memoria che sono intrinsecamente indipendenti dalla posizione. Qualsiasi istruzione di salto condizionato o BRA impiega il contatore di programma nell'indirizzamento con spostamento. Ovviamente i valori immediati non sono alterati dal trasferimento di un programma. Gli indirizzi fissi hanno valori definiti dal sistema e che pertanto non vengono modificati. Essi vengono solitamente definiti nel programma mediante la direttiva di uguaglianza (*EQUate:* EQU). Inoltre, il programmatore può usare indirizzi di memoria relativi al (PC) per garantire che il codice sia indipendente dalla posizione.

Come mostrato nella Tab. 9.6, la maggior parte delle istruzioni dell'MC68020 che specificano due operandi permettono che soltanto l'operando di sorgente possa essere specificato dal modo d'indirizzamento relativo al PC. Quindi l'istruzione

```
MOVE.W    X,Y
```

potrebbe avere soltanto X specificato come relativo al PC. La locazione della destinazione per BRA, Bcc, DBcc, JMP o JSR, tuttavia, può essere specificata da un modo relativo al PC. In sostanza, l'MC68020 non ammette che un operando passibile di modifica possa essere riferito dall'indirizzamento relativo al PC, che dai progettisti dell'MC68020 è considerato come un riferimento ad un'istruzione di programma, non a dati.[2] Le uniche eccezioni sono l'istruzione di test del bit (*Bit*

[2] Questa distinzione sarà spiegata nel cap. 13, quando saranno discusse le linee di codice di funzione.

TeST: BTST), di confronto immediato (*CoMPare Immediate:* CMPI), e di test (TST), poiché queste istruzioni non modificano l'operando di destinazione.

Tab. 9.5 *Riferimenti indipendenti dalla posizione.*

Categoria	Modo d'indirizzamento
Istruzioni di salto BRA Bcc DBcc	Relativo al PC con spostamento
Istruzioni immediate Logiche (ORI, ANDI, EORI) Aritmetiche (ADDI, SUBI, CMPI, ADDQ, SUBQ) Trasferimenti (MOVEQ)	Immediato
Riferimento assoluto a locazioni fisse	Assoluto lungo, assoluto corto
Riferimenti relativi alla memoria	Relativo al PC con spostamento, relativo al PC con indice

Tab. 9.6 *Indirizzamento relativo per istruzioni.*

Sorgente	Destinazione[1]
ADD, ADDA AND BFEXTS, BFEXTU, BFFFO, BFTST CHK, CHK2 CMP, CMPA, CMP2 cpRESTORE DIV, DIVSL, DIVU, DIVUL LEA MOVE, MOVEA MOVE TO CCR MOVE TO SR MOVEM MULS, MULU OR PEA SUB, SUBA	BRA, Bcc, DBcc BSR BTST CMPI JMP, JSR TST

Note:

1. La destinazione non viene modifcata da queste istruzioni.
2. Anche CALLM ammette l'indirizzamento relativo al PC. Questa istruzione è descritta nell'app. D.

Convenzoni di assemblatore e indirizzamento indipendente dalla posizione. Gli assemblatori possono presentare lievi differenze per quanto concerne la maniera in cui l'indirizzamento relativo è specificato dal programmatore. Come descritto nel cap. 5, la modalità d'indirizzamento relativa al PC viene richiamata dal modo:

(ETICH,PC)

dove ETICH è un'etichetta in un programma in linguaggio assembler. L'indirizzo viene generato aggiungendo uno spostamento (ETICH) al valore del PC quando viene eseguita un'istruzione che impiega questa modalità. Per esempio, l'istruzione:

```
MOVE.L      (DATA,PC),D1
```

trasferisce 32 bit dalla locazione indirizzata da (PC) + (DATI), col seguente risultato:

(D1) = ((PC) + DATO)

dove DATO è uno spostamento di 8, o 16 o 32 bit con segno. La seguente istruzione, che impiega un indirizzamento indiretto di memoria relativo al PC:

```
ADD.L       ([INDIR,PC]),D1
```

somma a (D1) l'operando puntato dal contenuto della locazione (PC) + INDIR, cioè:

(D1) = ((PC + INDIR)) + (D1)

Dunque viene "imposto" un indirizzamento relativo al PC quando il PC è esplicitamente specificato in qualsiasi modalità d'indirizzamento ammessa per l'istruzione in fase di codifica. Le istruzioni che consentono un indirizzamento relativo al PC sono elencate nella Tab. 9.6.

Nella maggior parte degli assemblatori è disponibile un'alternativa alla specificazione di "PC" in ciascun riferimento alla memoria in un'istruzione. La direttiva di assemblatore

```
OPT         PCO
```

produce l'indirizzamento relativo al PC su riferimenti a ritroso in un'etichetta. Con questa opzione, l'istruzione di esempio:

```
MOVE.W      ARRAY3,D0
```

trasferisce in D0 il valore di 16 bit contenuto nella locazione (PC) + ARRAY3, mentre il risultato

(D0)[W] = ((PC) + ARRAY3)[W]

viene trasferito il D0, purché ARRAY3 sia stato definito come un'etichetta *prima* che l'istruzione MOVE fosse incontrata dall'assemblatore. Sono possibili anche riferimenti in avanti, mediante le direttive OPT PCS (assemblatore della Motorola) oppure OPT PCF (assemblatore della Quelo). Per ulteriori dettagli, si dovrebbe consultare il manuale per l'utente dell'assemblatore specifico.

Esempio 9-4

La Fig. 9.3 presenta due subroutine che confrontano l'uso di indirizzi rilocabili e relativi. Nella prima routine, che inizia dall'etichetta ABSMOVE, l'indirizzo iniziale di un'area di buffer di byte nella memoria viene definito come BUFABS, dopo il salvataggio degli indirizzi. Al valore #BUFABS sarebbe assegnata la locazione assoluta $0016 se il programma fosse caricato alla locazione $0000. Comunque, l'assemblatore contrassegna come "rilocabile" l'etichetta BUFABS nell'istruzione:

```
MOVE.L    #BUFABS,A1
```

per indicare che l'editor di collegamento (*linkage editor*) o il programma caricato (*loader*) dovrà fornire l'indirizzo corretto prima che il programma venga caricato nella memoria. Ciò è indicato dalla tabella di simboli interna dell'assemblatore. Per comodità, il valore rilocabile è indicato anche nel listato della Fig. 9.3, nella colonna che contiene il linguaggio-macchina. Prima che il programma sia caricato ad un indirizzo di caricamento, si dovrebbe aggiungere un valore di rilocazione all'offset ($16) nell'istruzione MOVE che contiene il riferimento a BUFABS. Se la routine contenesse una direttiva ORG, l'etichetta BUFABS rappresenterebbe un indirizzo assoluto, per cui la routine non potrebbe essere rilocata.

La routine indipendente dalla posizione inizia dall'etichetta RELMOVE. Si noti l'impiego della direttiva SECTION per l'azzeramento del contatore di locazione dell'assemblatore. Ciò indica che questa sezione del programma è indipendente dalle precedenti istruzioni. Quando il programma viene caricato nella memoria ed eseguito, l'istruzione LEA trasferisce in A1 l'indirizzo (PC) + BUFREL, col risultato:

(A1) = (PC) + BUFREL

Qui BUFREL = $12 come offset nell'istruzione LEA. Per esempio, se il programma viene caricato alla locazione $10000, l'indirizzo è:

$10006 + $12 = $10018

```
                          1.          TTL      FIGURA 9.3
                          2.          LLEN     100
                          3.  *
                          4.  *  ASSEMBLAGGIO ASSOLUTO E ASSEMBLAGGIO RELATIVO
                          5.  *
                          6.  *  TRASFERIMENTO DI UN BLOCCO DI DATI
                          7.  *
                          8.  *       INPUT:  (D1.W) = NUMERO DI BYTE DA TRASFERIRE
                          9.  *               (A0.L) = INDIRIZZO DEL BLOCCO
                         10.  *
                         11.  *       OUTPUT: IL BUFFER E' CARICATO CON BYTE
                         12.  *
                         13.  *
                         14.  *
                         15.  *       SEZIONE DI CODICE RILOCABILE
                         16.  *
                         17.
0'000000  48E7 40C0      18.  ABSMOVE MOVEM.L D1/A0-A1,-(SP)  ;SALVA I REGISTRI
0'000004  227C'00000016  19.          MOVE.L  #BUFABS,A1      ;LEGGE L'INDIRIZZO DEL BUFFER
0'00000A  12D8           20.  LOOP1   MOVE.B  (A0)+,(A1)+     ;TRASFERISCE UN BYTE NEL BUFFER
0'00000C  5341           21.          SUBQ.W  #1,D1           ;DECREMENTA IL CONTATORE
0'00000E  66 FA          22.          BNE     LOOP1           ;CONTINUA FINCHE'
                         23.  *                               ;  (D1) = 0
0'000010  4CDF 0302      24.          MOVEM.L (AS)+,D1/A0-A1  ;RIPRISTINA I REGISTRI
0'000014  4E75           25.          RTS
0'000016  <64>           26.  BUFABS  DS.B    100             ;PREDISPONE IL BUFFER (ASSOLUTO)
                         27.  *
                         28.  *       SEZIONE DI CODICE INDIPENDENTE DALLA POSIZIONE
                         29.  *
                         30.
1'000000                 31.          SECTION 1
                         32.  *
1'000000  48E7 40C0      33.  RELMOVE MOVEM.L D1/A0-A1,-(SP)  ;SALVA I REGISTRI
1'000004  43FB 0170      34.          LEA     (BUFREL,PC),A1  ;RELATIVO AL PC
          00000012
1'00000C  12D8           35.  LOOP2   MOVE.B  (A0)+,(A1)+     ;TRASF. UN BYTE; RIFERIMENTO
                         36.  *                               ;  RELATIVO ALLA DESTINAZIONE
1'00000E  5341           37.          SUBQ.W  #1,D1           ;DECREMENTA IL CONTATORE
1'000010  66 FA          38.          BNE     LOOP2           ;CONTINUA FINCHE' (D1) = 0
1'000012  4CDF 0302      39.          MOVEM.L (SP)+,D1/A0-A1  ;RIPRISTINA I REGISTRI
1'000016  4E75           40.          RTS
                         41.  *
1'000018  <64>           42.  BUFREL  DS.B    100             ;PREDISPONE IL BUFFER (RELATIVO)
                         43.  *
1'00007C                 44.          END
```

Fig. 9.3 *Assemblaggio rilocabile e relativo.*

poiché il valore $10006 del PC punta due byte oltre l'istruzione LEA, quando tale istruzione impiega effettivamente il valore di (PC) per calcolare l'indirizzo da trasferire in A1. La successiva istruzione MOVE.B trasferisce un byte da una locazione puntata da A0 in un array definito da BUFREL. Quindi l'indirizzo di destinazione per l'istruzione MOVE.B è un indirizzo relativo al PC.

La subroutine definita nella Sezione 1 del programma può dunque essere spostata nella memoria ed eseguita senza l'impiego di un editor di collegamento o di un programma caricatore per assegnare gli indirizzi a BUFREL o a LOOP2. Al contrario, le istruzioni rilocabili ma dipendenti dalla posizione nella sezione rilocabile non potrebbero essere spostate nella memoria ed eseguite, a meno che un editor di collegamento o un programma caricatore non abbia assegnato un indirizzo assoluto a BUFABS. L'etichetta LOOP1 nella sezione rilocabile non richiede alcuna assegnazione d'indirizzo, poiché è definita dal modo d'indirizzamento relativo al PC dall'istruzione BNE.

9.2.2 Indirizzamento di registro di base

Nella discussione del codice indipendente dalla posizione, il contatore di programma era usato come un *indirizzo di base* a cui era aggiunto uno spostamento ed un valore di indice per individuare un operando nella memoria. I modi d'indirizzamento indiretto di registro d'indirizzo dell'MC68020 possono essere usati per ottenere un indirizzamento di *registro di base* impiegando uno qualsiasi dei registri d'indirizzo. Mentre viene eseguita l'istruzione che impiega l'indirizzamento con registro di base, la locazione dell'operando viene calcolata sommando l'offset all'indirizzo di base.

Nei programmi dell'MC68020, l'indirizzamento di registro di base è tipicamente impiegato per accedere a dati in un array o in una struttura simile o per passare ad una subroutine l'indirizzo di base di un'area di dati. Questo metodo d'indirizzamento è particolarmente quando la posizione relativa di un elemento di dati può essere individuata da uno spostamento o da un valore di indice ma l'indirizzo iniziale della struttura non è noto al momento dell'assemblaggio.

Se gli elementi contenuti in una tabella di valori nella memoria rappresentano indirizzi, allora si possono impiegare le modalità d'indirizzamento indiretto di memoria definite nel cap. 5. L'indirizzo di base di una tabella di indirizzi dovrebbe essere specificato come il contenuto di un registro d'indirizzo, prima che venga eseguito il programma che utilizza tali modalità d'indirizzamento. Di solito, è il linkage editor o il programma caricatore del computer che assegna l'indirizzo di base dopo che un programma è stato rilocato nella memoria. Gli operandi sono contenuti in locazioni indirizzate dalla tabella. Questi operandi non sono rilocabili se s'impiega il modo d'indirizzamento indiretto di memoria preindicizzato. Se invece s'impiega il modo indiretto di memoria postindicizzato, si potrebbe impiegare il registro indice che agisce sulla tabella di operandi come indirizzo di base di una tabella rilocabile di operandi.

Esempio 9-5

La Fig. 9.4 illustra la possibilità di segmentare la memoria in blocchi mediante l'indirizzamento di registro di base. I registri A1, A2, e A3 indirizzano segmenti diversi. I riferimenti di memoria in un programma che impiega A1 riguardano il primo segmento, ed un valore di indice potrebbe essere aggiunto per accedere ad una locazione specifica. Similmente, gli altri registri di indirizzo potrebbero essere usati per localizzare dati o istruzioni in altri segmenti. Se un segmento venisse spostato, il sistema operativo dovrebbe ricaricare il nuovo indirizzo iniziale nei registri d'indirizzo appropriati.

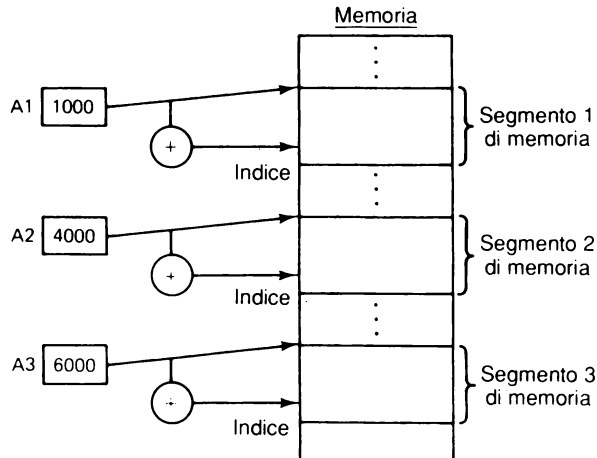



Fig. 9.4 *Assemblaggio rilocabile e relativo.*

9.2.3 Tecniche di programmazione di codice indipendente dalla posizione

Se un programma contiene soltanto valori immediati, indirizzi assoluti fissati dal sistema, e riferimenti relativi al PC ad operandi di memoria, allora tale programma sarà staticamente indipendente dalla posizione. L'indipendenza dinamica dalla posizione si ottiene normalmente tramite un indirizzamento con registro di base o mediante una conversione hardware degli indirizzi. In questi casi, sono richieste opportune routine del sistema operativo ed un'unità hardware di gestione della memoria per imporre l'indipendenza dinamica dalla posizione.

Un programma indipendente dalla posizione non può contenere riferimenti a valori di dati in indirizzi assoluti, a meno che tali valori non siano contenuti ad indirizzi prefissati, definiti dal sistema. I valori di dati trasferiti tra programma dovrebbero essere definiti in registri del processore o nello stack di sistema. Queste tecniche saranno esplorate ulteriormente nel par. 9.4, allorché saranno discusse le subroutine.

La Tab. 9.7 riassume le tecniche impiegate per la scrittura di codice staticamente indipendente dalla posizione, utilizzando l'indirizzamento relativo al PC per riferimenti di programma o per certi riferimenti di dati in un programma. Anche i metodi per il passaggio dei dati saranno presentati e definiti più in dettaglio nel par. 9.4.

Tab. 9.7 *Tecniche di programmazione indipendenti dalla posizione.*

Impiego	Tecnica
Riferimenti di programma	BRA, Bcc, DBcc. Modi d'indirizzamento relativi al PC espliciti od opzioni di assemblatore per produrre un indirizzamento relativo al PC.
Riferimenti di dati (sorgente)	Valori immediati. Indirizzi assoluti fissati dal sistema. Indirizzamento relativo al PC.
Riferimenti di dati (destinazione)	Istruzione LEA per trasferire un indirizzo relativo al PC in un registro d'indirizzo; quest'ultimo viene usato per far riferimento all'operando.
Passaggio di dati tra programmi	Registri del processore. Trasferimento di stack. Codifica in linea. Frame di stack.

Nota: i riferimenti di dati fanno sì che la CPU indichi un riferimento allo spazio di programma se s'impiega un modo d'indirizzamento relativo al PC. Ciò sarà spiegato nel cap. 13.

ESERCIZI

9.2.1

Si consideri il semplice segmento di programma:

```
        ORG     0
START   MOVE.W  PRIMO,D1
        ADD.W   SECONDO,D1
        MOVE.W  D1,RISULT
        RTS
PRIMO   DC.W    1
SECONDO DS.W    1
RISULT  DS.W    1
        END
```

Questo programma tenta di calcolare:

(RISULT) = (PRIMO) – (SECONDO)

ma non funzionerà. Inoltre, il programma non può essere spostato nella memoria dalla locazione 0000. Si corregga il programma in modo che funzioni ovunque nella memoria.

9.2.2

Si mostri che il seguente programma è staticamente indipendente dalla posizione:

```
         ORG       0
INIZ     LEA       *+0,A0              ;legge il PC
         ADDA.L    #(DATI-INIZ),A0     ;riloca il puntatore
         MOVE.W    #19,D1              ;contatore
CICLO    ADD.W     (A0)+,D2            ;somma l'array
         DBF       D1,CICLO
         RTS
DATI     DS.W      20
         END
```

Si mostri che il puntatore all'array di dati (A0) contiene l'indirizzo appropriato di DATI dopo che il programma viene spostato. In molti assemblatori, s'impiega DBRA in luogo di DBF.

9.2.3

Si supponga che le istruzioni

```
LEA      Y(PC),A1
ADD.W    X(PC),(A1)
```

siano usate per creare del codice indipendente dalla posizione. Sono specificati indirizzi relativi sia per la sorgente X che per la destinazione Y. Dopo l'esecuzione dell'istruzione LEA, il registro A1 contiene l'indirizzo di Y, calcolato dal valore di (PC) più lo spostamento. Si dimostri che il programma non è dinamicamente indipendente dalla posizione. (*Suggerimento:* si consideri il caso in cui il programma sia spostato dopo l'esecuzione di LEA ma prima che venga eseguita l'istruzione ADD.)

9.2.4

Si confronti il modo d'indirizzamento indicizzato con quello di registro di base. Nell'MC68020 le due modalità d'indirizzamento sono identiche, per quanto concerne il codice di linguaggio-macchina, ma hanno scopi differenti. Si discutano gli impieghi di ciascuna modalità.

9.2.5

Si scriva un programma indipendente dalla posizione per sommare i valori in cinque locazioni riservate da una direttiva DS nel programma. Lo si provi scrivendo un altro programma che sposta il programma che somma i valori e lo esegua dopo che è stato spostato.

9.3 STRUTTURE DI DATI

I tipi di dati fondamentali per l'MC68020 sono interi con segno o senza segno, interi BCD e variabili booleane. Essi sono considerati tipi di dati fondamentali o primitivi, poiché le istruzioni dell'MC68020 sono disponibili per manipolarli direttamente. Per contro, le stringhe di caratteri devono essere create e manipolate da algoritmi che sono ideati dal programmatore. Queste rappresentano un tipo di dati non disponibile a livello di linguaggio assembler. La definizione di nuovi tipi di dati e le relazioni logiche che definiscono la loro organizzazione conducono alla studio delle *strutture di dati.*

Un *array* è un esempio di struttura di dati. Un array consiste di un insieme di elementi di un singolo tipo di dati, contenuti in locazioni contigue della memoria. Il termine "array" e "tabella" sono di solito considerati sinonimi. Negli esempi dei capitoli precedenti, si sono impiegati array di numeri e tabelle di indirizzi. In questo paragrafo i concetti saranno generalizzati sia per array unidimensionali che multidimensionali.

I modi d'indirizzamento indiretto di registro d'indirizzo e indiretto di PC sono impiegati per indirizzare array di operandi nella memoria. Si può usare una qualsiasi modalità d'indirizzamento indiretto della memoria per indirizzare una tabella di indirizzi nella memoria, che a sua volta individua un array di operandi. Quando s'impiega un registro indice per individuare un elemento in un array o in una tabella, si può usare l'istruzione CMP2 (*CoMPare register against bounds:* confronta registro con i confini), presentata in questo paragrafo, per garantire che il valore dell'indice non superi certi limiti. Le istruzioni CHK e CHK2 (*CHcK register against bounds:* verifica se registro nei confini) svolgono una funzione simile, ma esse non saranno discusse fino al cap. 11, poiché fanno scattare una trappola se il valore dell'indice supera i limiti.

La *lista concatenata* è un altro tipo di struttura di dati utile in molte applicazioni. Tale lista consente di memorizzare elementi di dati in locazioni di memoria non contigue, usando puntatori per indicare l'ubicazione dell'elemento successivo nella lista. L'MC68020, grazie alla sua estesa capacità di manipolazione degli indirizzi, è in grado di operare bene con programmi che impiegano liste concatenate. A questo punto, potrebbe servire un ripasso delle modalità d'indirizzamento descritte nel cap. 5.

9.3.1 Array unidimensionali

L'array unidimensionale è una struttura di dati che consiste di una raccolta di elementi in cui ciascun elemento è identificato univocamente da un valore di indice che corrisponde alla sua posizione nell'array. Poiché ogni elemento dell'array è del medesimo tipo di dati, la sua struttura è omogenea. In matematica, l'array unidimensionale di numeri è definito *vettore*, in cui la posizione di ciascun elemento è specificata da un "pedice". Se si sceglie arbitrariamente "1" come primo pedice, si ha:

$$V_1, V_2, \ldots, V_N$$

per un vettore di N elementi. La convenzione del FORTRAN richiede che un vettore inizi col pedice 1, mentre i suoi elementi sono indicati da:

V(1), V(2), ..., V(N)

dove V(i) indica l'indirizzo dell'elemento V_i

Altri linguaggi, come il Pascal o l'ALGOL, consentono di specificare arbitrariamente il primo indice. Nel linguaggio assembler, è disponibile una flessibilità completa. In questa discussione, V_i denoterà l'elemento *i*-esimo stesso, mentre V(i) sarà l'indirizzo dell'elemento *i*-esimo.

Calcolo dell'indirizzo. La memorizzazione sequenziale degli elementi in un array unidimensionale consente un facile riferimento a ciascun elemento in base al suo indirizzo. Se un array X di N elementi inizia dalla locazione X(1) e ciascun elemento occupa C byte, allora l'elemento *j*-esimo ha l'indirizzo:

$$X(j) = X(1) + C * (j - 1) \quad 1 \leq j \leq N$$

dove C è una costante.[3] La lunghezza in byte dell'array è:

$$\text{lunghezza} = (X(N) - X(1)) \times C$$

dove X(N) e X(1) rappresentano rispettivamente l'ultimo ed il primo indirizzo degli elementi. La Tab. 9.8 illustra il calcolo degli indirizzi per vari array, mentre la Fig. 9.5 illustra la relazione generale nella memoria.[4]

Tab. 9.8 *Indirizzamento di un array unidimensionale.*

Dimensione dell'elemento nella tabella	***C***	**Locazione *j*-esima**	**Lunghezza dell'array (byte)**
Byte	1	X(j) = X(1) + (j – 1)	*N*
Word	2	X(j) = X(1) + 2 * (j – j)	2*N*
Longword	4	X(j) = X(1) + 4 * (j – 1)	4*N*
Stringa (lunghezza fissa)	*k*	X(j) = X(1) + k * (j – 1)	*k* x *N*

Note:
1. *N* è il numero degli elementi, ciascuno lungo *C* byte.
2. L'intervallo dell'indice è $1 \leq j \leq N$. Il primo indirizzo è X(1), che contiene il valore X_1.

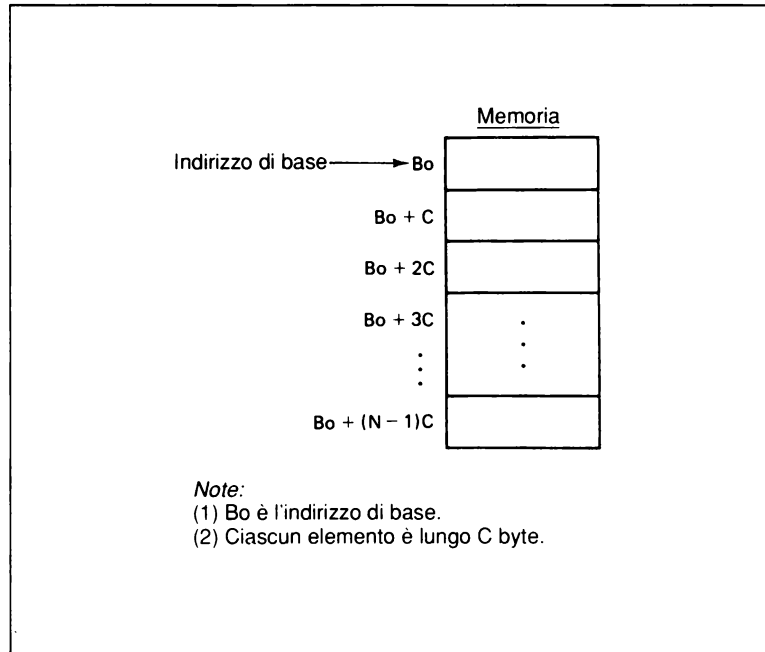
[3] Se X(0) è il primo indirizzo, allora l'elemento *j*-esimo ha l'indirizzo X(j) = X(0) + c x j, con j = 0, 1, 2, ..., N – 1.

[4] Non c'è alcun motivo per cui l'array non possa avere elementi con indici superiori che occupano locazioni inferiori nella memoria e quindi avere l'ordinamento fisico opposto a quello logico. In questo caso, le equazioni fornite in questo paragrafo richiederebbero una modifica.

Fig. 9.5
Memorizzazione di un array.

Memoria

Indirizzo di base ⟶ Bo	
Bo + C	
Bo + 2C	
Bo + 3C	.
.	.
.	.
Bo + (N − 1)C	

Note:
(1) Bo è l'indirizzo di base.
(2) Ciascun elemento è lungo C byte.

Indirizzamento di array con l'MC68020. L'indirizzamento degli elementi di un array si ottiene mediante le modalità d'indirizzamento indiretto o relativo dell'MC68020. Se l'indirizzo iniziale o di base di un array è contenuto in un registro d'indirizzo, si possono usare i modi indiretti per individuare gli elementi nell'array, come mostrato nella Tab. 9.9. Quando s'impiega il contatore di programma per effettuare l'indirizzamento relativo, un valore di spostamento o un valore di indice più spostamento è aggiunto al (PC) per far riferimento ad un elemento dell'array. Lo spostamento fisso è calcolato dall'assemblatore quando viene specificato l'indirizzamento relativo in un'istruzione; la memoria dell'array viene allocata mediante una direttiva *Define Storage* (DS) in un'istruzione etichettata.

Si può usare il modo d'indirizzamento con postincremento per far riferimento ad elementi dell'array in sequenza. Per elementi di byte, word o longword, la modalità d'indirizzamento con postincremento permette di esaminare e manipolare un elemento, dopodiché il registro d'indirizzo conterrà l'indirizzo dell'elemento successivo nell'array. L'istruzione di esempio

```
MOVE.W    (A1)+,D1
```

trasferisce il valore di 16 bit indirizzato da A1 in (D1)[15:0]. Dopodiché il registro A1 viene incrementato di 2 per puntare all'elemento successivo nell'array. Per un array i cui elementi cono memorizzati in locazioni decrescenti nella memoria a partire dall'indirizzo di base, la modalità di predecremento permette la scansione

Tab. 9.9 *Indirizzamento di array nell'MC68020.*

Modo d'indirizzamento dell'MC68020	Impiego tipico
Predecremento	Indirizzamento di elementi di byte, word o longword in sequenza decrescente.
Postincremento	Indirizzamento di elementi di byte, word o longword in sequenza crescente.
(PC) con spostamento o (An) con spostamento	Individuazione di un elemento in posizione fissa rispetto all'indirizzo di base.
(PC) con indice o (An) con indice	Individuazione di un elemento in posizione arbitraria mediante un registro indice.
(PC) con indice scalato o (An) con indice scalato	Individuazione di un byte (SCALA = 1), di una word (SCALA = 2), di una longword (SCALA = 4) o di una quadword (SCALA = 8) in un array; l'elemento viene individuato mediante indirizzamento indicizzato.
(PC) con spostamento di base e indice o (An) con spostamento di base e indice	Individuazione in un array di un elemento definito da uno spostamento di base e da un indirizzo di base contenuto in (PC) o in (An).

dell'array nell'ordine inverso.[5] A meno che non si provveda ad un'apposita programmazione, soltanto elementi di byte, word o longword possono essere indirizzati in modo sequenziale con queste modalità.

Si può impiegare il modo indiretto di registro d'indirizzo con spostamento per indirizzare un elemento specifico in un array. Il registro d'indirizzo contiene l'indirizzo di base dell'array, mentre lo spostamento specifica la posizione relativa dell'array come un offset. Questa modalità è spesso impiegata per confrontare gli elementi in array distinti.

[5] I modi di predecremento e postincremento sono utili per manipolare stack e code di byte, word o longword. Questi sono array *dinamici*, poiché la lunghezza varia con l'esecuzione del programma. Queste strutture sono ulteriormente discusse in vari riferimenti bibliografici relativi a questo capitolo, elencati nell'app. E.

Come esempio, si considerino due array con elementi di lunghezza di word. Se (A1) punta al primo elemento X(1), mentre (A2) punta a Y(1), allora la sequenza di istruzioni

```
MOVE.W    (4,A1),D1      ; (X(3))
MOVE.W    (4,A2),D2      ; (Y(3))
CMP.W     D1,D2          ; CONFRONTA: (Y(3)) – (X(3))
```

confronta Y_3 e Y_3 situati agli indirizzi Y(3) e X(3), rispettivamente. Si noti che il terzo elemento è individuato da un offset di quattro byte dall'indirizzo di base, in accordo con l'equazione d'indirizzamento per X(j) fornita in precedenza. L'offset nel modo indiretto e nel modo relativo al contatore di programma con spostamento non può essere modificato dopo che il programma è stato assemblato, per cui l'impiego di queste modalità non consente l'indicizzazione nell'array.

La flessibilità è ottenuta mediante il modo d'indirizzamento indiretto di registro d'indirizzo con indice o quello relativo al PC con indice. In queste modalità, l'indirizzo di base consiste di un valore di registro (registro d'indirizzo o PC) ed eventualmente di un valore di spostamento esteso di segno. L'indice nell'array può essere calcolato prima che sia utilizzato dalla valutazione di un'*espressione* di indice comunque complessa. Per esempio, nell'istruzione

```
MOVE.W    (0,A1,D1.W),D1
```

l'indirizzo di base è contenuto in A1, mentre il valore meno significativo di 16 bit in D1 contiene l'indice. Il valore in D1 potrebbe essere calcolato mediante qualsiasi espressione matematica prima di essere usato come indice. Se l'indice di 16 bit non fosse sufficiente, si potrebbe specificare un indice lungo di 32 bit, conformemente alla discussione di queste modalità d'indirizzamento fornite nel cap. 5.

Di solito s'impiega il modo diretto di registro d'indirizzo con indice e scalamento o il modo indiretto relativo al CP con indice e scalamento per indicizzare un array con elementi di byte, word, longword o quadword. Mentre il contenuto del registro indice viene modificato in un ciclo, il valore dell'indice è moltiplicato per 1, 2 4, o 8 prima dell'uso, a seconda del fattore di scala. Quindi l'istruzione

```
MOVE.W    (A1,D1.W*4),D2
```

con (D1)[W] = n, indirizza l'operando di sorgente alla locazione

$$(A1) + 4*n$$

che è l'n-esima longword nell'array, quando n = 0, 1, 2, ... definisce l'indice dell'array. In termini dell'array definito nella Fig. 9.5, n = 0 corrisponde a X(1), mentre N = N + 1 corrisponde all'elemento X(N), quando n è il valore dell'indice.

I modi indiretti di PC e di registro d'indirizzo consentono di aggiungere uno spostamento di base all'indirizzo di base nel PC o in An. Questo metodo è talvolta

impiegato dopo un'area d'intestazione in un array che contiene informazioni concernenti l'array. L'istruzione

```
MOVE.B      (4,A1,D1.W*8),D2
```

indirizzerebbe il primo elemento dell'array di operandi di sorgente, all'indirizzo

(A1) + 4

saltando così quattro byte d'informazione di intestazione all'inizio dell'array il cui indirizzo di base è definito da (A1). Naturalmente, un altro punto di vista è che l'istruzione MOVE.B in questione indirizza il quinto byte di un elemento di quadword definito, il cui primo byte è indirizzato come

(A1) + (D1)[W]*8

e trasferisce questo byte in D2. Quindi la funzione dell'offset dipende dall'applicazione. La Tab. 9.9 intende soltanto suggerire alcuni possibili impieghi delle varie modalità d'indirizzamento di un array di dati.

Esempio 9-6

La subroutine ERRMSG in Fig. 9.6 calcola l'indirizzo di un elemento specificato in un array consistente di cinque elementi, ciascuno lungo 13 byte. L'array di messaggi di errore inizia alla locazione TABLE ed un offset da tale indirizzo viene calcolato in base al valore del byte meno significativo di D0. L'indirizzo dell'elemento è calcolato moltiplicando il numero di errore per 13 e aggiungendo questo valore all'indirizzo di base in A0. L'equazione d'indirizzamento in questo esempio è:

TABLE(K) = TABLE + 13 X (k)

dove k = 0, 1, 2, 3 o 4 rappresenta il numero di errore. Se (D0).[B] non è un intero compreso tra 0 e 4, l'indirizzo riportato in A0 non è valido.

Esempio 9-7

La subroutine in Fig. 9.7 esegue un ordinamento "a bolle" di un array di valori di 8 bit. Al completamento, la routine lascia il valore massimo nella prima locazione, seguito dagli altri valori in ordine numerico decrescente. L'indirizzo iniziale, la cui locazione contiene il primo elemento dell'array, è fornito in A0, mentre D0 dovrebbe contenere N, il numero di elementi nell'array. L'ordinamento dell'array avviene confrontando gli elementi a partire dall'ultimo valore nell'array per ciascuna "passata".

```
                             1.          TTL     FIGURA 9.6
                             2.          LLEN    100
00010000                     3.          ORG     $10000
                             4.  *
                             5.  *   MESSAGGI DI ERRORE
                             6.  *
                             7.  *       INPUT:  (D0.B) = NUMERO DI ERRORE, TRA 0 E 4
                             8.  *
                             9.  *       OUTPUT: (A0.L) = INDIRIZZO DEL MESSAGGIO DI ERRORE CORRISP.
                            10.  *
00010000  2F01              11.  ERRMSG  MOVE.L  D1,-(SP)         ;SALVA IL REGISTRO
00010002  41F9 00010014     12.          LEA     TABLE,A0         ;LEGGE INDIRIZZO DI INIZIO TABELLA
00010008  323C 000D         13.          MOVE.W  #13,D1           ;13 BYTE PER STRINGA
0001000C  C0C1              14.          MULU.W  D1,D0
0001000E  D0C0              15.          ADDA.W  D0,A0            ;CALCOLA L'INDICE
00010010  221F              16.          MOVE.L  (SP)+,D1         ;RIPRISTINA IL REGISTRO
00010012  4E75              17.          RTS
                            18.  *
                            19.  *   DEFINISCE LA TABELLA DEI MESSAGGI DI ERRORE
                            20.  *
00010014  4F 56 45 52 46    21.  TABLE   DC.B    'OVERFLOW     '
          4C 4F 57 20 20
          20 20 20
00010021  55 4E 44 45 52    22.          DC.B    'UNDERFLOW    '
          46 4C 4F 57 20
          20 20 20
0001002E  53 55 42 53 43    23.          DC.B    'SUBSCRIPT    '
          52 49 50 54 20
          20 20 20
0001003B  5A 45 52 4F 20    24.          DC.B    'ZERO DIVIDE  '
          44 49 56 49 44
          45 20 20
00010048  55 4E 49 4D 50    25.          DC.B    'UNIMPLEMENTED'
          4C 45 4D 45 4E
          54 45 44
                            26.  *
00010055                    27.          END
```

Fig. 9.6 *Indirizzamento di un elemento di un array. (Esempio 9-6)*

```
                             1.          TTL     FIGURA 9.7
                             2.          LLEN    100
00010000                     3.          ORG     $10000
                             4.  *
                             5.  *   ORDINAMENTO DI UNA TABELLA DI VALORI DI 8 BIT
                             6.  *
                             7.  *       INPUT:  (A0.L) = INDIRIZZO DELLA TABELLA DA ORDINARE
                             8.  *               (D0.L) = NUMERO DI ENTRATE NELLA TABELLA (NUM)
                             9.  *
00010000  48E7 FE82         10.  SORT    MOVEM.L D0-D6/A0/A6,-(SP)       ;SALVA I REGISTRI
00010004  5380              11.          SUBQ.L  #1,D0
00010006  2600              12.          MOVE.L  D0,D3                   ;CONTATORE1 = NUM - 1
00010008  4201              13.  SORT10  CLR.B   D1                      ;FLAG = 0
0001000A  2800              14.          MOVE.L  D0,D4                   ;CONTATORE2 = NUM - 1
                            15.  *
0001000C  4DF0 4800         16.  SORT20  LEA     (0,A0,D4.L),A6
00010010  1C2E FFFF         17.          MOVE.B  (-1,A6),D6              ;LEGGE IL VALORE (CNT2 + 1)
00010014  B61C              18.          CMP.B   (0,A6),D6               ;SE ESSO >= VALORE (CNT2),
00010016  6C00 000C         19.          BGE     SORT30                  ;  ALLORA SALTA
0001001A  123C 00FF         20.          MOVE.B  #$FF,D1                 ;IMPOSTA IL FLAG
0001001E  1D56 FFFF         21.          MOVE.B  (0,A6),(-1,A6)          ;SCAMBIA I VALORI
00010022  1C86              22.          MOVE.B  D6,(0,A6)
                            23.  *
00010024  4A01              24.  SORT30  TST.B   D1                      ;SE FLAG /= 0
00010026  5384              25.          SUBQ.L  #1,D4                   ;  DECREMENTA CNT2
00010028  66 E2             26.          BNE     SORT20                  ;  E RICICLA
                            27.  *
0001002A  4A01              28.          TST.B   D1                      ;SE FLAG /= 0
0001002C  5383              29.          SUBQ.L  #1,D3                   ;  ALLORA DECREMENTA CNT1
0001002E  66 D8             30.          BNE     SORT10                  ;  E RICICLA
                            31.  *
00010030  4CDF 417F         32.          MOVEM.L (SP)+,D0-D6/A0/A6       ;RIPRISTINA I REGISTRI
00010034  4E75              33.          RTS
00010036                    34.          END
```

Fig. 9.7 *Esempio di ordinamento. (Esempio 9-7)*

Questa routine impiega (D0) per impostare due contatori in D3 e D4. Il ciclo più interno, che inizia dall'etichetta SORT20, usa il contatore in D4 per indicizzare l'array e reperire gli elementi da ordinare. Questo ciclo scambia gli elementi finché non si verifica la condizione di salto. Dopodiché, se c'è stato almeno uno scambio di elementi, tale condizione sarà indicata dal valore $FF del flag in D1. Il ciclo più esterno serve poi per esaminare gli elementi da scambiare ancora. Qualora il funzionamento di questa subroutine non risultasse chiaro dai commenti del programma, si possono trovare spiegazioni più dettagliate degli ordinamenti a bolle nei riferimenti bibliografici per questo capitolo, riportati nell'app. E alla fine del libro.

L'istruzione CMP2. L'istruzione CMP2 (*CoMPare register against bounds:* confronta registro con i confini) è usata per determinare se il contenuto di un registro rientra nell'intervallo numerico definito dai "confini" inferiore e superiore. Questa istruzione esegue la seguente verifica sul registro Rn:

confine inferiore ≤ (Rn) ≤ confine superiore

dove Rn è un arbitrario registro d'indirizzo o di dati. Come mostrato nella Tab. 9.10, i confini sono registrati nella memoria, a partire dalla locazione definita da un indirizzo effettivo. L'indirizzo effettivo punta alla locazione che contiene il confine inferiore in istruzioni della forma:

```
CMP2.<l>    <EA>,<Rn>
```

in cui <l> = B, W o L, mentre <EA> è definito da qualsiasi modalità d'indirizzamento della memoria, tranne il modo con predecremento o con postincremento. Il confine superiore è registrato dopo quello inferiore, nella successiva locazione di byte, word o longword. Quando viene eseguita l'istruzione CMP2, i codici di condizione C e Z vengono modificati per indicare il risultato. Quindi l'istruzione CMP2 è seguita tipicamente da un'istruzione Bcc della forma BEQ, BNE, BCC, BCS o BHI. Le condizioni per un salto sono definite nella Tab. 9.10(b).

Per confronti con segno o senza segno, il valore più piccolo che rappresenta il confine inferiore deve precedere il confine superiore nella memoria. Il valore può essere un intero di 8, 16 o 32 bit la cui lunghezza è determinata dall'apposita specificazione nell'istruzione CMP2. Se il registro Rn è un registro d'indirizzo, allora un confine di byte o di word viene esteso a 32 bit prima del confronto. Tutti i 32 bit di An sono impiegati per il confronto. Se il registro Rn è un registro di dati, soltanto la lunghezza specificata da <l> viene confrontata. Come esempio, l'istruzione

```
CMP2.L      $20000,D2
```

confronta il valore di 32 bit contenuto in D2 con i confini di longword a partire dalla locazione $20000. Il confronto è

($20000) ≤ (D2).L ≤ ($20004)

dove ($20000) contiene il confine inferiore mentre ($20004) contiene il confine superiore. L'intervallo di ciascun confine per interi senza segno è da $0000 0000 a $FFFF FFFF. L'intervallo per interi con segno è da $8000 0000 a $7FFF FFFF.

Tab. 9.10 *L'istruzione CMP2.*

(a) Sintassi dell'istruzione		
Sintassi	**Modalità d'indirizzamento**	**Operazione**

CMP2.<I> <EA>,<Rn>	*Sorgente:* modi di controllo
	
	Registro: Dn o An arbitrario
		Codici di condizione:
		
		IF lim.inf. < (Rn) < lim.sup.
		THEN C = {0}, Z = {0}
		ELSE
		IF (Rn) = lim.inf.
		o (Rn) = lim.sup.
		THEN Z = {1}, C = {0}
		OTHERWISE
		C = {1}, Z = {0}

(b) Condizioni di salto dopo l'istruzione CMP2		
Istruzione	**Codici di condizione**	**Condizione**
BEQ	Z = {1}	(Rn) è uguale a uno dei confini
BNE	Z = {0}	(Rn) non è uguale ad alcun confine
BCC	C = {0}	(Rn) è entro i confini
BCS	C = {1}	(Rn) è fuori dei confini
BHI	C = {0} e Z = {0}	(Rn) è entro i confini ma non è uguale a nessuno dei due

Note:
1. <I> = B, W o L.
2. Le modalità di controllo includono tutti i modi di riferimento alla memoria, tranne –(An) o (An)+.
3. Nella memoria:
   <EA> ← LIMITE INFERIORE
   <EA> + k ← LIMITE SUPERIORE
dove k = 1 (byte), 2 (word) o 4 (longword).
4. Il confine inferiore dev'essere algebricamente minore (o uguale) al confine superiore.
5. Se <An> è il registro da esaminare, gli operandi lunghi un byte o una word sono estesi di segno a 32 bit da CMP2.

Questa istruzione può essere usata per esaminare l'intervallo dei valori di un indice o di un indirizzo effettivo che viene calcolato durante l'esecuzione del programma. Durante il debugging del programma, si può impiegare l'istruzione CMP2 per rivelare errori di indicizzazione. In altre applicazioni, l'istruzione può essere usata per impedire ad un programma di accedere a dati situati all'esterno dello spazio di memoria assegnato ad esso. Tuttavia, tale controllo dell'indirizzamento è

affidato completamente alla responsabilità del programmatore. Quando il sistema operativo controlla l'indirizzamento in questa maniera, s'impiegano le istruzioni CHK e CHK2 poiché esse causano un'eccezione di trappola quando il valore di un registro fuoriesce dai confini. Queste istruzioni saranno discusse nel cap. 11.

Esempio 9-8

La subroutine nella Fig. 9.8 esamina l'intervallo del contenuto di 32 bit del registro D0. I confini inferiore e superiore devono essere specificati nei registri D1 e D2, rispettivamente. Il risultato della verifica dei confini è definito in D3: se (D3) = 0, il valore del registro in D0 era uguale ad uno dei confini; invece, se (D3) = 1, il valore del registro è interno ai confini; se (D3) = –1 allorché la subroutine restituisce il controllo al programma chiamante, il valore del registro non rientra nei confini specificati.

```
                        1.          TTL     FIGURA 9.8
                        2.          LLEN    100
00010000                3.          ORG     $10000
                        4.  *
                        5.  *   VERIFICA CHE (D0) RISPETTI I CONFINI INFERIORE E SUPERIORE
                        6.  *
                        7.  *       SUBROUTINE PER DETERMINARE SE
                        8.  *       INF(D1) <= (D0).L <= SUP(D2)
                        9.  *
                        10. *       INPUT:  (D1.L) = CONFINE INFERIORE
                        11. *               (D2.L) = CONFINE SUPERIORE
                        12. *
                        13. *       OUTPUT  (D3.L) = 0 SE (D0) COINCIDE CON UNO DEI DUE CONFINI
                        14. *                      = 1 SE (D0) RIENTRA NEI CONFINI
                        15. *                      =-1 SE (D0) FUORIESCE DAI CONFINI
                        16. *
                        17. *       DEV'ESSERE: CONFINE INFERIORE <= CONFINE SUPERIORE
                        18. *
00010000 2F09           19. BNDSCHK MOVE.L  A1,-(SP)        ;SALVA IL REGISTRO
00010002 23C1 00010032  20.         MOVE.L  D1,LOWER        ;MEMORIZZA I CONFINI
00010008 23C2 00010036  21.         MOVE.L  D2,UPPER
0001000E 43F9 00010032  22.         LEA     LOWER,A1        ; INDIRIZZO DI CONFINI
                        23. *
00010014 04D1 0000      24.         CMP2.L  (A1),D0         ;VERIFICA ENTRAMBI I CONFINI
                        25. *
00010018 6700 000C      26.         BEQ     EQUAL           ;SE Z={1}, UGUALE A UN CONFINE
                        27. *
0001001C 6500 000E      28.         BCS     OUTBNDS         ;SE C={1}, FUORI DAI CONFINI
00010020 7601           29.         MOVE.L  #1,D3           ;ALTRIMENTI, DENTRO I CONFINI
00010022 225F           30.         MOVE.L  (SP)+,A1
00010024 4E75           31.         RTS
                        32. *
00010026 7600           33. EQUAL   MOVE.L  #0,D3           ;UGUALE A UNO DEI DUE CONFINI
00010028 225F           34.         MOVE.L  (SP)+,A1
0001002A 4E75           35.         RTS
                        36. *
0001002C 76FF           37. OUTBNDS MOVE.L  #-1,D3          ;FUORI DAI CONFINI
0001002E 225F           38.         MOVE.L  (SP)+,A1
00010030 4E75           39.         RTS
                        40. *
                        41. *       RISERVA LOCAZIONI PER I CONFINI
                        42. *
00010032 00000001       43. LOWER   DC.L    1
00010036 00000001       44. UPPER   DC.L    1
0001003A                45.         END
```

Fig. 9.8 *Subroutine per la verifica dei confini di registro.*

9.3.2 Array bidimensionali

Una generalizzazione dell'array unidimensionale è l'array a più dimensioni. Gli elementi in un array multidimensionale sono specificati da più di un pedice, come mostrato nella ab. 9.11(a) per un array bidimensionale M x N. La Tab. 9.11(b) mostra un array 3 x 3 come esempio specifico. In questa notazione, M è il numero di righe orizzontali, e N è il numero di colonne verticali. L'elemento nella i-esima riga e nella j-esima colonna è designato come X_{ij}, dove $1 \leq i \leq M$ e $1 \leq j \leq N$. X(i,j) rappresenta l'indirizzo nella memoria dell'elemento X_{ij} nella forma X(riga, colonna). Un array M x N ha M x N elementi. L'indice di riga ha un intervallo di M valori, mentre l'indice di colonna ha un intervallo di N valori, indipendentemente dagli indici iniziali. Alcuni aspetti concernenti la strutturazione dei dati nella memoria riguardano il metodo di memorizzazione per riga e per colonna, come pure le tecniche necessarie per calcolare l'indirizzo di un elemento.

Tab. 9.11 *Array multidimensionali.*

(a) *Forma generale*	(b) *Esempio* 3 x 3
X_{11} $X_{12} \cdots$ X_{1N}	X_{11} X_{12} X_{13}
X_{21} $X_{22} \cdots$ X_{2N}	X_{21} X_{22} X_{33}
$\cdot$ $\cdot$ $\cdot$	X_{31} X_{32} X_{33}
$\cdot$ $\cdot$ $\cdot$	
$\cdot$ $\cdot$ $\cdot$	
X_{M1} $X_{M2} \cdots$ X_{MN}	

(c) *Memorizzazione di un maggiore di colonna di un array 3* x *3 a partire dalla locazione* $1000

INDIRIZZO (ESADECIMALE)		ELEMENTO DELL'ARRAY
X(1,1)	$1000	X_{11}
X(2,1)	$1002	X_{21}
X(3,1)	$1004	X_{31}
X(1,2)	$1006	X_{12}
X(2,2)	$1008	X_{22}
X(3,2)	$100A	X_{32}
X(1,3)	$100C	X_{13}
X(2,3)	$100E	X_{23}
X(3,3)	$1010	X_{33}

Memorizzazione dell'array. Se l'array X di dimensioni *M* x *N* è memorizzato sequenzialmente per righe in locazioni a partire dall'inizio X(1,1), come segue:

X(1,1), X(1,2), ..., X(1,N), X(2,1), ..., X(2,N), ..., X(M,N)

allora la memorizzazione è detta in forma di *maggiore di riga*. Una forma alternativa è quella di *maggiore di colonna*, con indirizzi successivi degli elementi:

X(1,1), X(2,1), ..., X(M,1), X(1,2), ..., X(M,2), ..., X(M,N)

come mostrato nella Tab. 9.11(c), che illustra la memorizzazione di un array 3 x 3 di word nella memoria dell'MC68020, a partire dalla locazione $1000. Questa forma di maggiore di colonna è richiesta nel FORTRAN standard e sarà impiegata negli esempi in questo sottoparagrafo.

Una volta che la forma di memorizzazione è stata scelta, gli indici per un elemento specifico possono essere calcolati in vari modi. Il *polinomio d'indirizzo* per un array *M* x *N* ha la forma:

$$X(i,j) = \text{indirizzo di base} + C_1 \times (j-1) + C_2 \times (i-1)$$

per un array bidimensionale il cui primo indirizzo è alla locazione X(1,1). Con una scelta opportuna delle costanti C_1 e C_2, il calcolo dell'indirizzo è immediato per un processore che disponga dell'istruzione di moltiplicazione.[6]

La Tab. 9.12 mostra i polinomi d'indirizzo per array con elementi di lunghezza *C* byte. L'indirizzo di X_{ij} in un array memorizzato nella forma di maggiore di colonna è:

$$X(i,j) = Bo + C \times [(i-1) + M \times (j-1)]$$

in cui:

$$1 \leq i \leq M \quad \text{e} \quad 1 \leq j \leq N$$

e Bo rappresenta l'indirizzo di base. Questo metodo d'indirizzamento è facilmente realizzabile con l'MC68020, usando l'indirizzamento indiretto con indicizzazione, se la selezione riguarda elementi in una riga (o colonna) fissa. Se entrambi gli indici possono variare, un indirizzo dev'essere calcolato separatamente ed aggiunto al valore dell'indirizzo effettivo calcolato con l'indirizzamento indicizzato.

---

[6] In alcuni casi, per risparmiare il tempo richiesto dalla moltiplicazione o per consentire l'allocazione dinamica (durante l'esecuzione) della memoria dell'array, sono impiegati dei metodi speciali d'indirizzamento. Questi includono l'impiego di un "vettore di caduta" che descrive le caratteristiche dell'array o un tipo di indirizzamento indiretto in cui gli indirizzi di riga e di colonna sono contenuti in un tabella. Questo argomento è sviluppato in vari riferimenti bibliografici realtivi a questo capitolo, riportati nell'app. E alla fine del libro.

Tab. 9.12 *Indirizzamento di un array multidimensionale.*

Memorizzazione dell'array	Indirizzo X(i,j)
Memorizzazione in forma di maggiore di colonna X(1,1), X(2,1), ...	Bo + C x [(i – 1) + M x (j – 1)]
Memorizzazione in forma di maggiore di riga X(1,1), X(1,2), ...	Bo + C x [(j – 1) + N x (i – 1)]

Note:
1. Bo è l'indirizzo di base dell'array X(i,j) con elementi di lunghezza *C* byte.
2. Gli intervalli degli indici sono i seguenti:
   riga: $1 \leq i \leq M$
   colonna: $1 \leq j \leq N$

Esempio 9-9

Quando si calcola l'indirizzo di un elemento in un array bidimensionale, due indici devono essere sommati all'indirizzo iniziale o di base. I modi d'indirizzamento indiretto di registro d'indirizzo con indice e relativo al PC con indice dell'MC68020 consentono due offset separati e calcolano l'indirizzo effettivo come:

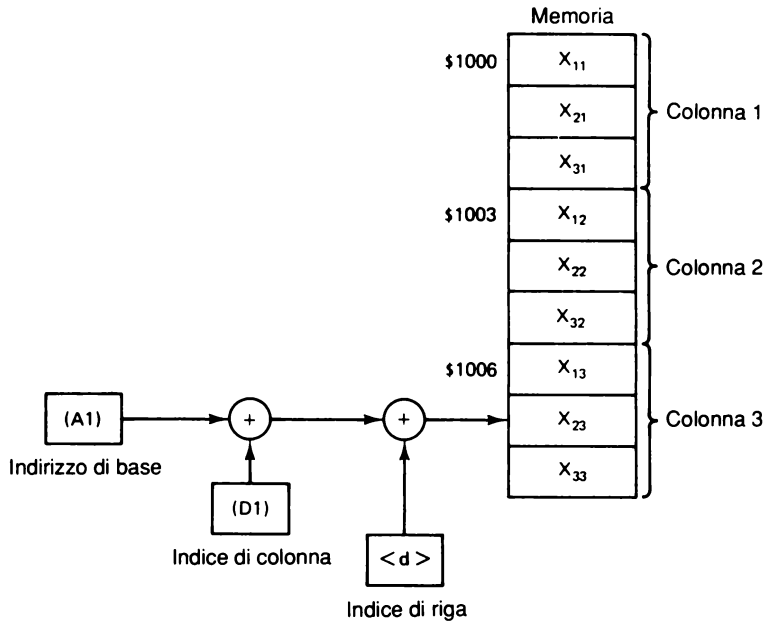
<EA> = (R) + (Rn) + <d>

Qui il registro R potrebbe essere un registro d'indirizzo nel modo indiretto oppure il contatore di programma per il modo relativo. Il registro indice Rn può essere un registro d'indirizzo o un registro di dati. La Fig. 9.9 illustra l'impiego di questo metodo d'indirizzamento per un array memorizzato in forma di maggiore di colonna. Se il numero di byte per elemento è 1, il calcolo dell'indirizzo diviene:

$$X(i,j) = Bo + [(i - 1) + M \times (j - 1)]$$

L'indirizzo di base ed uno degli indici possono essere contenuti in registri. Il secondo indice può essere specificato come l'offset nel modo d'indirizzamento indiretto di registro d'indirizzo con indice. Nella figura, l'offset seleziona la riga e rimane fisso. L'array 3 x 3 inizia dalla locazione esadecimale $1000, che è contenuta in A1, mentre il registro D1 contiene l'indice di colonna (j – 1) x M, che vale $0006. L'istruzione

```
MOVE.B      (1,A1,D1.L),D2
```

trasferisce l'elemento X_{23} al byte meno significativo di D2. Per selezionare A_{2j} da un'altra colonna, (D1) dev'essere modificato per indicare l'offset di colonna, che viene calcolato come 3 x (j – 1), con j = 1, 2 o 3.

Esempio: Carica X_{23} in D2

```
MOVE.B (1,A1,D1.L),D2 ;(D2)[7:0] ← ((A1)) + (D1) + 1)

(A1) = $0000 1000 (Base)
(D1) = $0000 0006 (Indice di colonna = (j – 1) * M)
<d> = 1           (Indice di riga = i – 1)
```

Fig. 9.9 *Indirizzamento di elemento fisso.*

Esempio 9-10

La subroutine nella Fig. 9.10 esegue una ricerca binaria in una tabella o in un array, per trovare una configurazione di bit lunga una word, il cui valore costituisce la cosiddetta "chiave" della ricerca. La tabella è composta da *M* elementi, ciascuno dei quali è lungo *N* byte. L'indirizzo iniziale della tabella viene fornito in A0, mentre la chiave da trovare è in (D0)[7:0]. La word meno significativa di D1 contiene la lunghezza (*N*) di ciascun elemento e (D2)[15:0] contiene il numero (*M*) di elementi nella tabella.

```
                           1.              TTL      FIGURA 9.10
                           2.              LLEN     100
00010000                   3.              ORG      $10000
                           4.    *
                           5.    *     SUBROUTINE PER RICERCA BINARIA IN UNA TABELLA
                           6.    *         INPUT:   (A0.L) = TABELLA DI RICERCA
                           7.    *                  (D0.B) = CHIAVE DI RICERCA NELLA LUNGHEZZA DI BYTE
                           8.    *                           DELLA TABELLA
                           9.    *                  (D1.W) = LUNGHEZZA IN BYTE DI CIASCUNA ENTRATA
                          10.    *                           NELLA TABELLA
                          11.    *                  (D2.W) = NUMERO DI ENTRATE NELLA TABELLA (FINE)
                          12.    *
                          13.    *         OUTPUT:  (A6.L) = INDIRIZZO DELL'ELEMENTO NELLA TABELLA
                          14.    *                           COL VALORE DELLA CHIAVE (O NULLO)
                          15.    *
00010000  48E7 FC00       16.    SEARCH    MOVEM.L D0-D5,-(SP)      ;SALVA REGISTRI IN STACK
00010004  5342            17.              SUBQ.W  #1,D2            ;INIZ : FINE (D2)
00010006  4283            18.              CLR.L   D3               ;       INIZIO (D3)
00010008  2C43            19.              MOVEA.L D3,A6            ;       VALORE DI USCITA
                          20.    *
0001000A  B642            21.    SER10     CMP.W   D2,D3            ;SE INIZIO >= FINE
0001000C  6E00 0028       22.              BGT     EXIT             ;  ALLORA ESCE
00010010  3803            23.              MOVE.W  D3,D4            ;  ALTRIMENTI CALCOLA
00010012  D842            24.              ADD.W   D2,D4            ;       INDICE = (INIZIO
00010014  E24C            25.              LSR.W   #1,D4            ;                + FINE)/2
                          26.    *                                  ;
                          27.    *
00010016  3A04            28.              MOVE.W  D4,D5            ;CALCOLA INDIRIZZO
00010018  CAC1            29.              MULU.W  D1,D5            ;INDICE IN TABELLA DI CHIAVE
                          30.    *                                  ;  DA ESAMINARE
0001001A  B030 5000       31.              CMP.B   (0,A0,D5),D0     ;SE CHIAVE >= ENTRATA
0001001E  6C00 0008       32.              BGE     SER20            ;  ALLORA SALTA PER MODIFICA
                          33.    *
00010022  5344            34.              SUBQ.W  #1,D4            ;  ALTRIMENTI PONE
00010024  3404            35.              MOVE.W  D4,D2            ;       FINE = INDICE - 1
00010026  60 E2           36.              BRA     SER10            ;  RIPROVA
                          37.    *
                          38.    *
00010028  6700 0008       39.    SER20     BEQ     SUCCESS          ;SE CHIAVE = ENTRATA DI TABELLA
                          40.    *                                  ;  ALLORA SALTA PER AGGIORNARE
                          41.    *                                  ;    L'USCITA
0001002C  5244            42.              ADDQ.W  #1,D4            ;  ALTRIMENTI
0001002E  3604            43.              MOVE.W  D4,D3            ;       INIZIO = INDICE + 1
00010030  60 D8           44.              BRA     SER10            ;       RIPROVA
                          45.    *
                          46.    *
00010032  4DF0 5000       47.    SUCCESS   LEA     (0,A0,D5.W),A6   ;SALVA INDIRIZZO DI USCITA
00010036  4CDF 003F       48.    EXIT      MOVEM.L (SP)+,D0-D5      ;RIPRISTINA I REGISTRI
0001003A  4E75            49.              RTS
0001003C                  50.              END
```

Fig. 9.10 *Routine di ricerca.*

Se la ricerca va a buon fine, l'indirizzo del valore della chiave è riportato in A6; altrimenti, A6 conterrà il valore zero.Poiché si effettua una ricerca binaria, si presume che i dati siano ordinati numericamente nella tabella oggetto della ricerca.

Esempio 9-11

La Fig. 9.11 illustra la struttura di dati per un array bidimensionale nella memoria. L'indirizzo iniziale di ciascuna colonna di lunghezza M è contenuto in una tabella di indirizzi puntata da A0. L'array è definito come:

$$X(I,J) \qquad I = 1, 2, ..., M; \quad J = 1, 2, ..., N$$

Fig. 9.11
Indirizzamento indiretto di un array multi-dimensionale

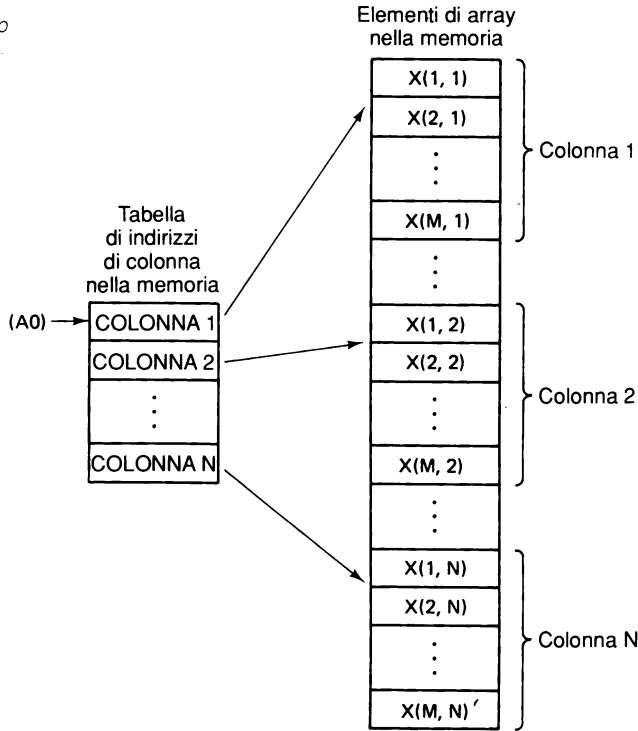
Elementi di array nella memoria

Tabella di indirizzi di colonna nella memoria

(A0) → COLONNA 1, COLONNA 2, ..., COLONNA N

X(1, 1), X(2, 1), ..., X(M, 1) — Colonna 1

X(1, 2), X(2, 2), ..., X(M, 2) — Colonna 2

X(1, N), X(2, N), ..., X(M, N) — Colonna N

L'array è memorizzato in forma di maggiore di colonna.
Gli elementi sono designati come X(riga,colonna) o X(I,J)
con I = 1, 2, ..., M, e J = 1, 2, ..., N.

e consiste di M righe e N colonne di elementi di lunghezza di word (16 bit). La subroutine della Fig. 9.12 calcola l'indirizzo dell'elemento X(I,J) come:

$$X(I,J) = ((A0) + 4 \times (J - 1)) + 2 \times (I - 1)$$

in cui D0 contiene l'indice di riga I, mentre D1 contiene l'indice di colonna J. Dopodiché, il contenuto della locazione X(I,J) viene trasferito in D2. L'impiego della tabella di indirizzi di colonna elimina il calcolo richiesto per determinare l'offset rispetto ad una certa colonna. Ciò rende più veloce l'esecuzione, però richiede la presenza di una tabella di N indirizzi nella memoria.

9.3.3 Liste concatenate

Quando si opera con un array, il successore dell'elemento indirizzato viene individuato aggiungendo una costante all'indirizzo dell'elemento presente. Per esempio, in un array unidimensionale, si ha:

```
                           1.         TTL     FIGURA 9.12
                           2.         LLEN    100
00010000                   3.         ORG     $10000
                           4.  *
                           5.  *   CALCOLA L'INDIRIZZO DELL'ELEMENTO X(I,J) DELL'ARRAY
                           6.  *   E TRASFERISCE L'ELEMENTO A (D2).W
                           7.  *
                           8.  *   X(I,J) E' UN ARRAY DI ELEMENTI DI LUNGHEZZA DI WORD
                           9.  *   LE CUI COLONNE SONO INDIRIZZATE DA UNA TABELLA
                          10.  *   PUNTATA DA A0.
                          11.  *
                          12.  *       INPUT:  (A0.L) = INDIRIZZO BASE DELLA TABELLA
                          13.  *                        DI INDIRIZZI DI COLONNA
                          14.  *               (D0.W) = INDICE I DI RIGA:     I = 1,2,...,M
                          15.  *               (D1.W) = INDICE J DI COLONNA:  J = 1,2,...,N
                          16.  *
                          17.  *       OUTPUT: (D2.W) = ELEMENTO X(I,J)
                          18.  *               (A2.L) = INDIRIZZO DELL'ELEMENTO X(I,J)
                          19.  *
                          20.  *       NOTA:   SE I E J NON SONO ENTRAMBI NELL'INTERVALLO
                          21.  *               APPROPRIATO, D2 E A2 CONTERRANNO VALORI ERRATI.
                          22.  *
00010000  48E7 C040       23.  TWOD   MOVEM.L D0/D1/A1,-(SP)      ;SALVA I REGISTRI
00010004  5340            24.         SUBQ.W  #1,D0               ;OFFSET A RIGA
00010006  5341            25.         SUBQ.W  #1,D1               ;OFFSET A INDIR. DI COLONNA
                          26.  *
00010008  43F0 1511       27.         LEA     ([A0,D1.W*4]),A1    ;INDIRIZZO DI COLONNA
                          28.  *                                  ; DELLA J-MA COLONNA
0001000C  45F1 0200       29.         LEA     (A1,D0.W*2),A2      ;INDIRIZZO DI X(I,J)
00010010  3412            30.         MOVE.W  (A2),D2             ;(D2)[W] = X(I,J)
                          31.  *
00010012  4CDF 0203       32.         MOVEM.L (SP)+,D0/D1/A1      ;RIPRISTINA I REGISTRI
00010016  4E75            33.         RTS
00010018                  34.         END
```

Fig. 9.12 *Subroutine per calcolare l'indirizzo di un array con l'indirizzamento indiretto di colonna. (Esempio 9-11)*

$$X(j + i) = X(j) + C$$

in cui C è il numero di byte occupati da ciascun elemento. Gli elementi, che sono ordinati in progressione, occupano blocchi contigui di memoria, come evidenziato nella Fig. 9.13. Al confronto, la *lista concatenata* è una struttura di dati che non richiede la memorizzazione contigua dei suoi elementi. La lista concatenata sarà trattata a livello introduttivo in questo sottoparagrafo. Le operazioni avanzate su liste di questo tipo, tra cui la gestione dello spazio di memoria occupato dalla lista e l'ordinamento dei suoi elementi, sono discusse in vari riferimenti bibliografici relativi a questo capitolo, elencati nell'app. E alla fine del libro.

Il disegno della Fig. 9.14(a) illustra un esempio di lista concatenata di cinque elementi. Ogni elemento nella lista contiene, oltre all'elemento di dati, anche un puntatore (indirizzo) — denominato *collegamento* — che punta all'elemento successivo nella lista. La lista mostrata ha un collegamento *unidirezionale*, poiché ogni elemento può avere un solo successore. Inoltre, la lista in questione non è ordinata, poiché gli elementi di dati non si succedono in ordine numerico.

Fig. 9.13 *Memorizzazione di un array sequenziale.*

Note:
(1) Bo è l'indirizzo di base.
(2) Ciascun elemento è lungo C byte.

(a) *Esempio di lista concatenata.*

(b) *Allocazione della memoria.*

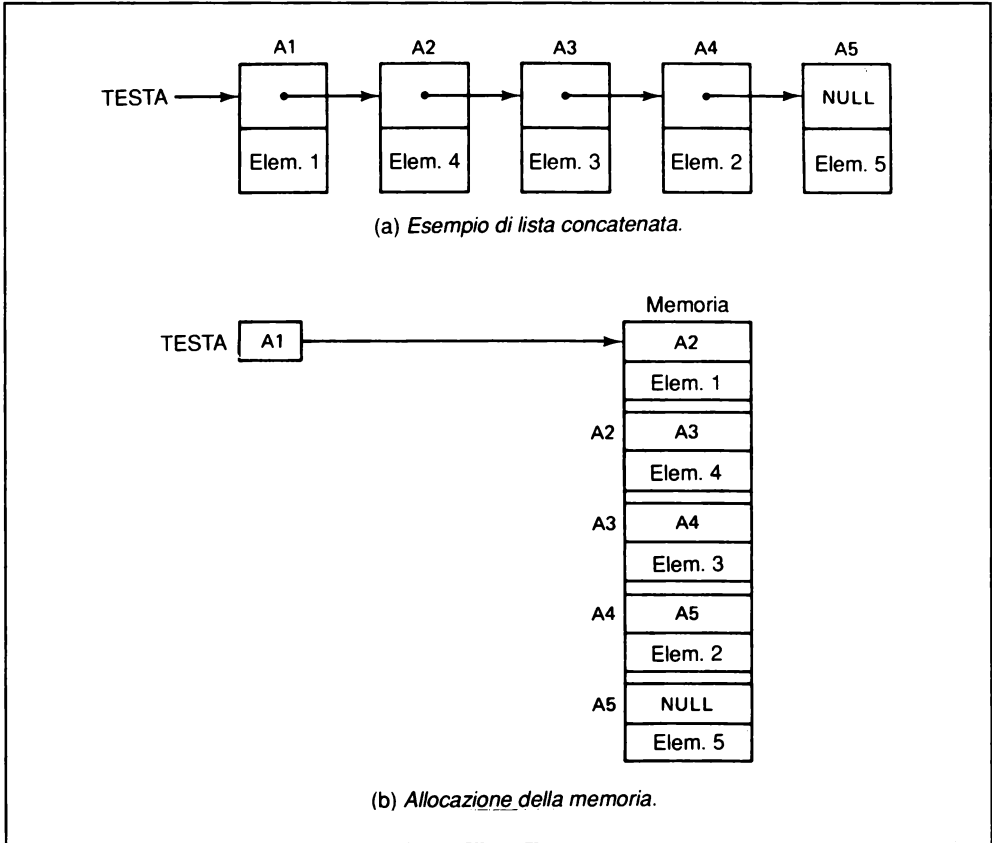
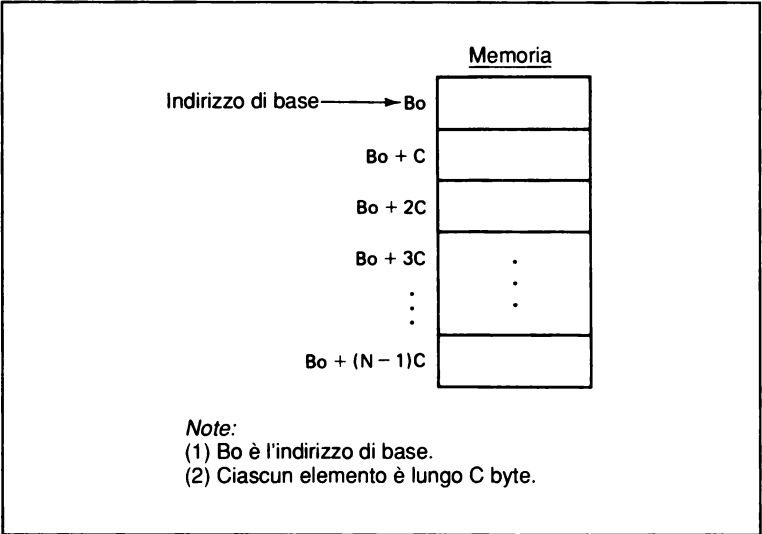



Fig. 9.14 *Allocazione per una lista concatenata non ordinata.*

Il puntatore alla lista è memorizzato all'indirizzo TESTA. L'ultimo elemento, contraddistinto dal numero 5 nella figura, contiene un simbolo speciale, designato come NULL, che indica la fine della lista. Per esempio, si potrebbe usare un valore NULL uguale a zero in programmi dell'MC68020, poiché nessun elemento di dati sarebbe memorizzato alla locazione $0000. Se la locazione TESTA contiene il valore NULL, allora la lista è vuota. La Fig. 9.14(b) illustra il modo in cui la lista in questione potrebbe essere contenuta nella memoria.

Esempio 9-12

La Fig. 9.15 illustra una subroutine che crea una lista concatenata in un'area libera della memoria con indirizzo iniziale AVAIL (da *available*, cioè: disponibile). I nodi o elementi nella lista saranno inizializzati per contenere 10 entrate. Il ciclo che inizia dall'etichetta LINK (collegamento) calcola l'indirizzo del nodo successivo nella lista e poi memorizza tale indirizzo come collegamento a quel nodo. Al termine del ciclo, il primo collegamento che era stato memorizzato sarà riscritto con un valore NULL.

```
                            1.          TTL      FIGURA 9.15
                            2.          LLEN     100
00010000                    3.          ORG      $10000
                            4.  *
    * 00000000              5.  NULL    EQU      0                  ;PREDISPONE VALORE PER NULL.
                            6.  *
                            7.  *   CREA UNA LISTA CONCATENATA
                            8.  *
                            9.  *       INPUT:   AVAIL E' L'INDIRIZZO DI UN BLOCCO LIBERO DI MEMORIA
                           10.  *                HEAD E' IL PUNTATORE ALL'INIZIO DELLA LISTA
                           11.  *
                           12.  *       OUTPUT:  I LEGAMI SONO MEMORIZZATI NEL BLOCCO IN AVAIL;
                           13.  *                HEAD PUNTA AL PRIMO NODO NELLA LISTA
                           14.  *
00010000  48E7 C0C0        15.  LNKLST  MOVEM.L  D0-D1/A0-A1,-(SP)       ;SALVA I REGISTRI
00010004  41F9 0001002E    16.          LEA      AVAIL,A0
0001000A  23C8 0001002A    17.          MOVE.L   A0,HEAD            ;IMPOSTA HEAD PER PUNTARE AD AVAIL
00010010  7008             18.          MOVE.L   #8,D0              ;IMPOSTA BYTE/NODO
00010012  720A             19.          MOVE.L   #10,D1             ;IMPOSTA NUMERO DI NODI
                           20.  *
00010014  2248             21.  LINK    MOVE.L   A0,A1              ;INDIRIZZO DI NODO IN A1
00010016  D1C0             22.          ADD.L    D0,A0              ;DETERMINA IL NODO SUCCESSIVO
00010018  2288             23.          MOVE.L   A0,(A1)            ;MEMORIZZA LEGAME AL NODO SUCC.
0001001A  5381             24.          SUBQ.L   #1,D1              ;DECREMENTA IL NUMERO DI NODI
0001001C  66 F6            25.          BNE      LINK               ;  FINCHE' IL CONTO NON E' ZERO
                           26.  *
0001001E  22BC 00000000    27.          MOVE.L   #NULL,(A1)         ;RISCRIVE L'ULTIMO LEGAME
                           28.  *
00010024  4CDF 0303        29.          MOVEM.L  (SP)+,D0-D1/A0-A1       ;RIPRISTINA I REGISTRI
00010028  4E75             30.          RTS
                           31.  *
0001002A  <4>              32.  HEAD    DS.L     1                  ;PUNTATORE ALLA CIMA DELLA LISTA
0001002E  <50>             33.  AVAIL   DS.L     20                 ;BLOCCO DI MEMORIA PER LA LISTA
                           34.  *
0001007E                   35.          END
```

Fig. 9.15 *Subroutine per creare una lista concatenata.*

Le istruzioni CAS e CAS2 per liste concatenate. L'MC68020 dispone di due istruzioni per la gestione di liste concatenate. L'istruzione CAS (*Compare And Swap:* confronta e scambia) può essere usata per inserire o cancellare un elemento da una singola lista concatenata. Essa non offre alcun particolare vantaggio di programmazione, tuttavia impiega un ciclo indivisibile di accesso alla memoria (lettura-modifica-scrittura) che è indivisibile per aggiornare la testa di una lista. L'istruzione CAS2 può servire ad aggiornare i puntatori d'indirizzo in una lista doppiamente concatenata, mediante un ciclo di memoria indivisibile. Tali istruzioni saranno trattate nel par. 12.6, allorché saranno discussi i sistemi multiprocessore.

ESERCIZI

9.3.1 Si scriva una subroutine per determinare il seno di un angolo espresso in gradi, quando viene specificato un angolo da 0 a 360 gradi. Si supponga che sia già disponibile una tabella con indirizzo iniziale SENO e che essa contenga valori di 16 bit per il seno di angoli da 0 a 89 gradi. Se l'angolo è maggiore di 90 gradi, il suo seno dovrà essere calcolato servendosi delle identità trigonometriche. [*Nota:* il valore potrebbe essere calcolato usando l'espansione in serie di Taylor di SIN(X), cioè:

$$SIN(X) = X - (X^3/3!) + (x^5/5!) - \cdots$$

Il metodo della serie fornirà il valore del seno con qualunque approssimazione si desideri, ma risulta molto più lento del metodo di ricerca tabellare, se la tabella ha una risoluzione sufficiente. Si crei una tabella abbreviata di valori di seno e si provi la routine.]

9.3.2 Si scriva una subroutine per "pulire" un array tridimensionale *senza* calcolare il polinomio d'indirizzo tridimensionale. Si supponga che l'array sia memorizzato in forma di maggiore di colonna e che la subroutine riceva come parametri l'indirizzo iniziale, le dimensioni dell'array (M x N x O) e il numero di byte in ciascun elemento.

9.3.3 Si scriva una routine per moltiplicare due matrici 2 x 2.

9.3.4 Si scriva il polinomio d'indirizzo per un array k-dimensionale se I_1, I_2, I_3, ..., I_k sono gli indici e L_1, L_2, L_3, ..., L_k sono le lunghezze.

9.3.5 Si scriva una subroutine per rimuovere un elemento dalla cima della lista concatenata creata nell'esempio 9.12. Che cosa accade se la lista è vuota [cioè, se (TESTA) = NULL]?

9.3.6 Si supponga che:

(D3) = $ABCD FFFC
(A0) = $10000
($10000) = $FFF0
($10002) = $0040

prima che sia eseguita l'istruzione

```
CMP2.W    (A0),D3
```

Qual è il codice di condizione risultante dopo l'esecuzione dell'istruzione?

9.3.7 Si modifichi la subroutine dell'esempio 9.8 per trasformarla in un segmento di macrocodice che permetta di specificare Rn e la lunghezza dei confini (B, W o L) all'atto della chiamata della macro.

9.4 IMPIEGO DELLE SUBROUTINE E PASSAGGIO DEGLI ARGOMENTI

L'impiego di *subroutine* o procedure è una tecnica di programmazione importante per creare programmi modulari, in cui ciascuna subroutine svolge un compito specifico entro il programma complessivo. Il metodo per trasferire il controllo tra il programma chiamante e la subroutine è denominato *collegamento di subroutine*. Nell'MC68020, la chiamata ad una subroutine è eseguita dall'istruzione

```
JSR    <SUBR>
```

la cui prima azione consiste nell'inserire in cima allo stack di sistema l'indirizzo di ritorno contenuto nel programma chiamante. Dopodiché, il controllo viene trasferito alla subroutine all'indirizzo <SUBR>. L'indirizzo può essere specificato mediante una qualsiasi modalità d'indirizzamento di controllo dell'MC68020. Quindi il trasferimento del controllo avviene molto semplicemente nell'MC68020. Quando devono essere scambiati dei dati tra il programma chiamante e la subroutine, si può ricorrere a vari metodi per il trasferimento delle informazioni. Il metodo viene selezionato nella fase di progettazione del programma e questa scelta costituisce una parte importante del progetto del programma stesso.

Le informazioni richieste dalla subroutine sono definite in termini di *parametri*, che consentono alla subroutine di gestire casi generali anziché limitarsi ad operare su valori specifici. Ogni chiamata ad una subroutine permette di specificare valori diversi, denominati *argomenti* per i parametri. Alcuni riferimenti tipici di una subroutine FORTRAN sono mostrati nella Fig. 9.16. Questa subroutine è denominata SUBR ed ha i parametri A, B e C. Essa può essere chiamata specificando vari argomenti, purché questi siano dati del medesimo tipo (interi, in virgola mobile, ecc.) dei parametri che compaiono nella definizione della subroutine. I nomi o valori degli argomenti sono arbitrari. I nomi simbolici per gli argomenti nell'esempio sono effettivamente assegnati dal compilatore. I valori specifici 1.0 e 3.0 nella seconda chiamata possono essere sostituiti per trarre vantaggio dalla flessibilità del linguaggio FORTRAN. Nel linguaggio assembler, la distinzione tra valori effettivi e gli indirizzi degli argomenti è importante.

```
-----------------------------------------------------
!                         !                          !
! Programma chiamante     ! Subroutine               !
!                         !                          !
!-------------------------!--------------------------!
!                         !                          !
!                         ! SUBROUTINE SUBR (A, B, C)!
!      .                  !    .                     !
!      .                  !    .                     !
!      .                  !    .                     !
! CALL SUBR (X, Y, Z)     !    .                     !
!      .                  !    .                     !
!      .                  ! RETURN                   !
!      .                  ! END                      !
! CALL SUBR (1.0, 3.0, RESULT) !                     !
!      .                  !                          !
!      .                  !                          !
!      .                  !                          !
! CALL SUBR (A(1), W, ANS)!                          !
!      .                  !                          !
!      .                  !                          !
!      .                  !                          !
! END                     !                          !
!                         !                          !
!-------------------------!--------------------------!
```

FIg. 9.16 *Uso di una subroutine del FORTRAN.*

I "meccanismi" della definizione dei parametri e della trasmissione degli argomenti alla subroutine sono molto più complessi nel linguaggio assembler. Gli argomenti possono essere contenuti in registri del processore, nello stack di sistema o in locazioni fisse della memoria. Inoltre, si possono usare le istruzioni LINK e UNLK dell'MC68020 per creare uno o più *frame di stack* all'atto della chiamata di una subroutine. Tale frame è un blocco di memoria riservato sullo stack, che contiene l'indirizzo di ritorno, gli argomenti e le eventuali variabili locali. Si possono realizzare subroutine ricorsive o rientranti tramite questo metodo di gestione dei dati nella subroutine.

9.4.1 Passaggio degli argomenti alle subroutine

I parametri, che definiscono gli argomenti da trasferire tra una subroutine ed il programma chiamante, possono essere valori di dati, indirizzi o combinazioni di entrambi. Quando dev'essere trasferito soltanto un piccolo numero di argomenti, essi vengono passati direttamente tra i programmi in registri del processore. Nel caso in cui sono passate molte variabili o quando si fa riferimento ad una struttura di dati quale un array, viene trasferito l'indirizzo del gruppo di variabili o della struttura di dati. Le distinzioni in questo caso sono piuttosto vaghe in molti programmi scritti in linguaggio ad alto livello, per quanto concerne le operazioni del compilatore. Nel linguaggio assembler, la distinzione tra valori ed indirizzi è importante poiché il metodo di passaggio dei parametri determina il modo in cui si accede agli argomenti.

Varie tecniche impiegate per il passaggio di valori o indirizzi tra i programmi sono elencate nella Tab. 9.13. Il programma chiamante definisce la sequenza di chiamata, compresa la definizione degli argomenti da trasferire alla subroutine. La subroutine accede quindi agli argomenti per l'elaborazione ed eventualmente riporta dei valori o indirizzi al programma chiamante. Gli argomenti passati alla subroutine sono definiti come *parametri d'ingresso*. I risultati sono valori o indirizzi che corrispondono ai *parametri di uscita* per la subroutine. Naturalmente, una combinazione delle tecniche elencate nella Tab. 9.13 potrebbe essere usata quando è definito un insieme complesso di parametri d'ingresso/uscita.

Tab. 9.13 *Metodi di passaggio degli argomenti.*

Tipo	Descrizione	Commenti
Registro	La routine chiamante carica i valori o gli indirizzi nei registri prestabiliti.	Il numero di parametri è limitato. Dinamico.
Stack	La routine chiamante pone i valori o gli indirizzi in cima allo stack.	L'indirizzo di ritorno dev'essere salvato durante l'elaborazione e ripristinato prima del ritorno.
Aree di parametri	Vengono definite le aree di memoria che contengono i valori o gli indirizzi.	Statico, se le aree sono definite in fase di assemblaggio. Dinamico, se l'indirizzo di base dell'area viene passato in un registro.
In-linea	I valori o gli indirizzi sono memorizzati in seguito alla chiamata. La subroutine calcola l'ubicazione dei parametri.	Statico

Trasferimento di registro. Il metodo più semplice per il passaggio degli argomenti consiste nell'impiegare l'insieme di registri dell'MC68020. I valori dei dati possono essere passati in uno qualunque degli otto registri di dati. Similmente, si possono utilizzare i registri d'indirizzo per il passaggio di indirizzi che possono puntare a valori di dati o contenere gli indirizzi iniziali di strutture di dati. Il metodo di passaggio in registri offre i vantaggi di semplicità, di ridotte esigenze di memoria e di minimo tempo di esecuzione. Il numero degli argomenti che possono essere passati è limitato al numero di registri disponibili, che sono 15 per l'MC68020. Il progettista della subroutine e quello della routine chiamante devono solo mettersi d'accordo sui registri utilizzati per il passaggio degli argomenti. Per esempio, la sequenza di istruzioni

```
MOVE.W     VALORE,D1,     ; DATI
MOVEA.L    ADDTAB,A1      ; PUNTATORE
LEA        TESTA,A2       ; INDIRIZZO DI TESTA
JSR        SUBR
```

predispone un valore (VALORE) di 16 bit in D1, il puntatore d'indirizzo nella locazione ADDTAB in A1, e l'indirizzo TESTA in A2. La subroutine SUBR potrà quindi accedere direttamente ai valori nei registri per svolgere la propria funzione.

Trasferimento di stack. Uno stack può essere impiegato per passare gli argomenti facendo sì che la subroutine chiamante inserisca valori o indirizzi sullo stack prima della chiamata. Per rappresentare il puntatore di stack, si potrebbe usare uno stack privato, che può essere definito nei programmi dell'MC68020 mediante uno dei registri d'indirizzo A0, A1, ..., A6. I valori sono inseriti nello stack usando nel programma chiamante la modalità d'indirizzamento con predecremento o con postincremento. Il prelievo degli argomenti dallo stack per trasferirli alla subroutine consente di accedere a valori o a indirizzi. Per uno stack privato, la modifica del puntatore di stack durante l'esecuzione non influisce sulle operazioni del sistema e viene gestita a discrezione del programmatore. La sequenza per predisporre A1 come puntatore di stack e per passare due valori potrebbe essere la seguente:

```
LEA        STACKP,A1      ; INDIRIZZO DELLO STACK IN A1
MOVE.W     VAL1,(A1)+     ; INSERISCE IL PRIMO VALORE
MOVE.W     VAL2,(A1)+     ; INSERISCE IL SECONDO VALORE
JSR        SUBR
```

dove STACKP è il FONDO dello stack, che cresce verso locazioni a indirizzi più alti nella memoria. La subroutine accede ai valori mediante la sequenza:

```
MOVE.W     -(A1),D2  ; SECONDO VALORE
MOVE.W     -(A1),D1  ; PRIMO VALORE
```

se l'obiettivo è il caricamento dei valori nei registri di dati. Il puntatore di stack A1 contiene ora il suo valore originale STACKP. In questo esempio, l'argomento nel registro A1 conteneva l'indirizzo di una struttura di dati (lo stack) nella memoria.

Quando s'impiega lo stack di sistema per il passaggio degli argomenti, l'indirizzo di ritorno si trova in cima allo stack allorché inizia l'esecuzione della subroutine. Questo valore dev'essere rimosso prima che la subroutine possa estrarre gli argomenti dallo stack. Quando l'elaborazione della subroutine è completata, l'indirizzo di ritorno dev'essere posto nuovamente in cima allo stack, prima che l'istruzione RTS possa essere eseguita. La sequenza di chiamata per questo metodo di passaggio dei parametri potrebbe essere la seguente:

```
PEA       INDIR          ; INSERISCE L'INDIRIZZO
MOVE.W    VAL1,-(SP)     ; INSERISCE IL VALORE
JSR       SUBR
```

Dapprima viene posto sullo stack l'indirizzo INDIR, poi il valore nella locazione VAL1 e infine l'indirizzo di ritorno. Poiché (PC) è in cima allo stack, esso può essere salvato e successivamente ripristinato prima del ritorno. La sequenza per svolgere questa funzione potrebbe essere:

```
MOVE.L    (SP)+,A1       ; SALVA (PC) TEMPORANEAMENTE
MOVE.W    (SP)+,D1       ; PRENDE IL DATO
MOVEA.L   (SP)+,A2       ; PRENDE L'INDIRIZZO
          .
          .              ; ELABORAZIONE
          .
MOVE.L    A1,-(SP)       ; RIPRISTINA (PC)
RTS
```

Poiché questo metodo è solitamente impiegato da un programma che opera nel modo di utente, il puntatore di stack attivo è USP. La modifica dei puntatori di stack non interferisce con l'elaborazione delle interruzioni e con simili operazioni di sistema che utilizzano il puntatore dello stack di supervisore (SSP).

Locazioni di memoria per gli argomenti. Quando dev'essere passato un gran numero di parametri, si può predisporre un'opportuna *area di parametri* nella memoria. Tale area contiene, in una sequenza prestabilita, i valori o gli indirizzi a cui la subroutine potrà accedere dopo che ad essa sarà stato passato l'indirizzo iniziale dell'area. La stessa area potrebbe essere usata da diverse subroutine che richiedono parametri differenti, a patto che le sue dimensioni siano sufficientemente grandi da contenere il massimo numero di argomenti.

Un altro impiego di questo metodo è comune nei sistemi che hanno subroutine in ROM. L'area di parametri nella memoria RAM (*Random Access read/write Memory:* memoria di lettura/scrittura ad accesso casuale) è definita in accordo coi requisiti del sistema e l'indirizzo viene passato alle subroutine da utilizzare come indirizzo di base nell'accesso agli argomenti.[7]

---

[7] Un'altra versione, impiegata quando sono specificate aree "comuni" in FORTRAN, definisce l'indirizzo del'area di parametri sia per il programma chiamante che per la subroutine in fase di compilazione.

Il programma chiamante potrebbe predisporre un'area di parametri nel modo seguente:

```
                MOVE.W  VAL1,AREAPARM       ; MEMORIZZA DATO
                MOVE.W  VAL2,AREAPARM + 2   ; SECONDA WORD
                   .
                   .
                   .
                MOVE.W  VAL5,AREAPARM +8    ; QUINTA WORD
                LEA  AREAPARM,A1            ; INSERISCE INDIRIZZO
                JSR  SUBR
                   .
                   .
                   .
AREAPARM        DS.W    5                   ; AREA RISERVATA
                END
```

in cui cinque word sono definite come parametri. La subroutine potrebbe accedere ai valori usando l'indirizzamento indiretto con spostamento. Per esempio, l'istruzione:

```
MOVE.W    (6,A1),D1
```

trasferisce il quarto valore in D1. Sono possibili molte varianti per definire l'area di parametri nella memoria.

Codifica in-line. Un altro metodo di passaggio di valori ad una subroutine è quello di codificare i valori dopo la chiamata della subroutine. Questo metodo, noto come *codifica in-linea*, definisce dei valori degli argomenti che sono costanti e che non cambieranno dopo l'assemblaggio. Questi valori possono essere definiti da direttive DC poste dopo la chiamata. Si consideri la sequenza di istruzioni:

```
JSR       SUBR
DC.W      1          ; ARGOMENTO IN-LINEA
```

I 32 bit di (PC) sono inseriti nello stack di sistema dalla chiamata della subroutine. Questo indirizzo punta alla locazione dell'*argomento* nella sequenza di istruzioni. La sequenza di istruzioni riportata di seguito potrebbe essere eseguita dalla subroutine per caricare l'argomento nella word meno significativa di D1 e per far sì che l'indirizzo di ritorno posto in cima allo stack punti alla word situata oltre quel valore:

```
MOVEA.L   (A7),A0    ; PRENDE IL VALORE DI PC
MOVE.W    (A0)+,D1   ; PRENDE L'ARGOMENTO E INCREMENTA
MOVE.L    A0,(A7)    ; INSERISCE IL NUOVO INDIRIZZO DI RITORNO
          .
          .          ; ELABORAZIONE
          .
RTS
```

La prima istruzione carica il (PC) in A0 dallo stack. Successivamente il registro A0 indirizza l'argomento ed è incrementato di 2 dopo che D1 è stato caricato. Dopo che A0 è stato incrementato, esso punterà all'istruzione del programma chiamante che segue l'argomento in-linea. L'istruzione successiva nella subroutine inserisce l'indirizzo di ritorno corretto sullo stack, sovrascrivendo il valore già salvato dall'istruzione JSR. L'istruzione RTS è impiegata qui per ripristinare (PC) e restituire il controllo al programma chiamante. Il riferimento ad A7 indica il puntatore dello stack di sistema: USP o SSP, a seconda che il programma operi nel modo di utente o in quello di supervisore, rispettivamente.

9.4.2 Frame di stack

Una delle principali considerazioni nel progetto di subroutine riguarda il concetto di *trasparenza*. In parole semplici, allorché una subroutine ha terminato la propria esecuzione, essa non dovrebbe avere alcun effetto visibile, a parte quello definito dal suo legame col programma chiamante. Per esempio, una subroutine non dovrebbe modificare i valori contenuti nei registri, a meno che qualche registro non sia impiegato per il passaggio dei risultati. Nei precedenti esempi di programmi, questa condizione era soddisfatta salvando nello stack, subito dopo l'entrata nella subroutine, i contenuti dei registri che essa doveva utilizzare; tali valori venivano poi ripristinati prima di tornare al programma chiamante. L'indirizzo di ritorno veniva salvato e ripristinato automaticamente dalle istruzioni JSR e RTS.

L'impiego dello stack di sistema per salvare e ripristinare l'indirizzo di ritorno e i contenuti dei registri utilizzati entro la subroutine garantiva che i dettagli dell'operato della subroutine fossero trasparenti per il programma chiamante. Qualora una subroutine stessa avesse effettuato una chiamata ad un'altra subroutine, l'impiego dello stack per la memorizzazione temporanea dei contenuti di registri da parte di ciascuna subroutine e per ciascun indirizzo di ritorno permetteva un siffatto annidamento delle subroutine senza difficoltà. Questo concetto di utilizzazione dello stack per memorizzare temporaneamente i dati durante l'esecuzione della subroutine può essere esteso mediante la definizione di un *frame di stack*.

Il frame di stack è un blocco di memoria nello stack che viene usato per gli indirizzi di ritorno, i parametri d'ingresso, i parametri di uscita e le variabili locali. È l'area dello stack a cui una subroutine accede durante la sua esecuzione. Le variabili locali sono quei valori utilizzati durante l'esecuzione della subroutine e che non vengono ritrasmessi alla routine chiamante. Un contatore di ciclo, ad esempio, il cui valore cambia ad ogni iterazione entro la subroutine, potrebbe essere definito come una variabile locale. Ad ogni chiamata alla subroutine, questa può accedere un nuovo insieme di parametri, variabili locali e indirizzi di ritorno, utilizzando la tecnica del frame di stack. Se la subroutine viene chiamata prima di giungere al completamento, i valori nel frame di stack non saranno distrutti.

I frame di stack dell'MC68020. Nei sistemi in multiprogrammazione, vari compiti (*task*) indipendenti possono usare la medesima subroutine. Per esempio,

Tab. 9.14 *Operazioni di LINK e UNLK.*

Istruzione	Sintassi	Operazione
Collega	LINK.<l$_1$> <An>,#<spost>	1. (SP) ← (SP) – 4; ((SP)) ← (An) 2. (An) ← (SP) 3. (SP) ← (SP)+ <spost>
Scollega	UNLK <An>	1. (SP) ← (An) 2. (An) ← ((SP)); (SP) ← (SP) + 4

Note:

1. <spost> è un intero esteso di segno di 32 bit o di 16 bit. Uno spostamento negativo è specificato per allocare l'area di stack.
2. <l$_1$> = W o L.

al sistema possono essere connessi due terminali CRT distinti ma che condividono la medesima routine di I/O. Allorché il sistema operativo commuta il controllo tra i due terminali, è possibile che la routine di I/O di uno di essi venga interrotta e che il controllo sia passato temporaneamente all'altro terminale. Tutti i dati associati col primo terminale utilizzato dalla routine di I/O devono essere salvati in modo che, quando il primo terminale riacquista il controllo, l'esecuzione della routine di I/O possa riprendere dal punto in cui era rimasta. Tale utilizzazione richiede routine *rientranti*, in cui nessun dato nell'area di memoria del programma stesso cambia durante l'esecuzione. Qualsiasi valore che cambia viene posto sullo stack per la memorizzazione. Quindi il programma o codice è completamente distinto dai dati su cui opera. Un caso speciale è la routine *ricorsiva*, che chiama sé stessa e che quindi è autorientrante. Il frame di stack permette di creare facilmente routine rientranti e ricorsive.

Il frame di stack viene creato dal programma chiamante e dalla subroutine mediante le istruzioni LINK e UNLK dell'MC68020. La sintassi e le operazioni di queste istruzioni sono mostrate nella Tab. 9.14. L'accesso alle variabili sullo stack da parte della subroutine si ottiene tramite gli offset (cioè, mediante indicizzazione) da un registro di base denominato *puntatore di frame*. Sebbene il valore del puntatore di stack possa cambiare con l'inserimento o il prelievo degli elementi, il puntatore di frame non varia durante l'esecuzione della subroutine.

Esempio 9-13

La Fig. 9.17 illustra una possibile sequenza nel programma chiamante ed il funzionamento della subroutine. La Fig. 9.18 illustra il contenuto dello stack per questo esempio. Sono possibili molte varianti a seconda dell'applicazione. Nel caso mostrato, la routine chiamante riserva dapprima *N* byte sullo stack per gli argomenti che dovranno essere riportati dalla subroutine. Dopodiché un valore d'ingresso ed un indirizzo sono inseriti in cima allo stack.

```
                                1.          TTL     FIGURA 9.17
                                2.          LLEN    100
00010000                        3.          ORG     $10000
                                4.  *
                                5.  *   PROGRAMMA CHIAMANTE
                                6.  *
  = 00000008                    7.  N       EQU     8               ; 8 BYTE PER OUTPUT
  = 00000008                    8.  M       EQU     8               ; 8 BYTE PER VARIABILI LOCALI
                                9.  *       .
                                10. *       .
00010000 DFFC FFFFFFF8          11.         ADD.L   #-N,SP          ;AREA DI OUTPUT
00010006 2F39 0001001E          12.         MOVE.L  ARG,-(SP)       ;ARGOMENTO DI INPUT
0001000C 4879 00010022          13.         PEA     X               ;INDIRIZZO X DI INPUT
00010012 4E39 000100EA          14.         JSR     SUBR
00010018 508F                   15.         ADD.L   #8,SP           ;SALTA OLTRE INPUT
0001001A 221F                   16.         MOVE.L  (SP)+,D1        ;LEGGE INPUT
0001001C 241F                   17.         MOVE.L  (SP)+,D2
                                18. *       .
                                19. *       CONTINUA L'ELABORAZIONE COME RICHIESTO
                                20. *       .
                                21. *       .
                                22. *
                                23. *       (FINE DELL'ELABORAZIONE DEL PROGRAMMA PRINCIPALE)
                                24. *
0001001E 01234567               25. ARG     DC.L    $01234567       ;ARGOMENTO DA PASSARE
00010022 <C8>                   16. X       DS.B    200             ;TAB. IL CUI INDIRIZZO E' PASSATO
                                27. *
                                28. *   SUBROUTINE
                                29. *
000100EA 4E51 FFF8              30. SUBR    LINK    A1,#-M          ;SALVA VECCHIO FP
                                31. *       .
                                32. *       .
000100EE 2379 00010112          33.         MOVE.L  LOCAL1,(-4,A1)  ;SALVA VARIABILI LOCALI
         FFFC
000100F6 2379 00010116          34.         MOVE.L  LOCAL1,(-8,A1)
         FFF8
                                35. *       .
                                36. *       .
000100FE 52A9 FFFC              37.         ADD.L   #1,(-4,A1)      ;CAMBIA VARIABILE LOCALE
00010102 2469 0008              38.         MOVEA.L (8,A1),A2       ;LEGGE X
                                39. *       .
                                40. *       .
00010106 2379 0001011A          41.         MOVE.L  OUTPUT1,(16,A1) ;INSERISCE UN OUTPUT IN STACK
         0010
                                42. *       .
                                43. *       .
0001010E 4E59                   44.         UNLK    A1              ;RIPRISTINA SP E RITORNA
00010110 4E75                   45.         RTS
                                46. *
00010112 98765432               47. LOCAL1  DC.L    $98765432       ;VARIABILI LOCALI
00010116 87654321               48. LOCAL2  DC.L    $87654321
0001011A 41 42 43 44            49. OUTPUT1 DC.L    'ABCD'          ;VALORE DI OUTPUT
0001011E                        50.         END
```

Fig. 9.17 *Operazioni di un programma per la creazione di un frame di stack.*

L'istruzione JSR inserisce l'indirizzo di ritorno e trasferisce il controllo alla subroutine. A questo punto, lo stack contiene *N* byte di spazio per il risultato, il contenuto di 32 bit della locazione ARG, l'indirizzo X e l'indirizzo di ritorno.

La subroutine esegue dapprima l'istruzione LINK per creare un frame di stack e definire il puntatore di frame. Questa istruzione salva il valore di <An> sullo stack e sostituisce <An> col valore del puntatore di stack, usando A1 in questo esempio. Il puntatore di frame punterà così al *fondo* dell'area locale per la subroutine. Poi lo spostamento viene aggiunto al puntatore di stack cosicché (SP) punterà alla locazione situata *M* byte più in basso nella memoria.

(libera)
SP
(a) *Prima*

SP Indirizzo di ritorno
X
(ARG)
Area di OUTPUT
(b) *Dopo JSR*

(libera)
SP Area locale
FP Vecchio (A1)
Indirizzo di ritorno
X
(ARG)
Area di OUTPUT
(c) *Dopo LINK*

(libera)
SP Indirizzo di ritorno
X
(ARG)
(OUTPUT 1)
(OUTPUT 2)
(d) *Dopo UNLK*

SP X
(ARG)
(OUTPUT 1)
(OUTPUT 2)
(e) *Dopo RTS*

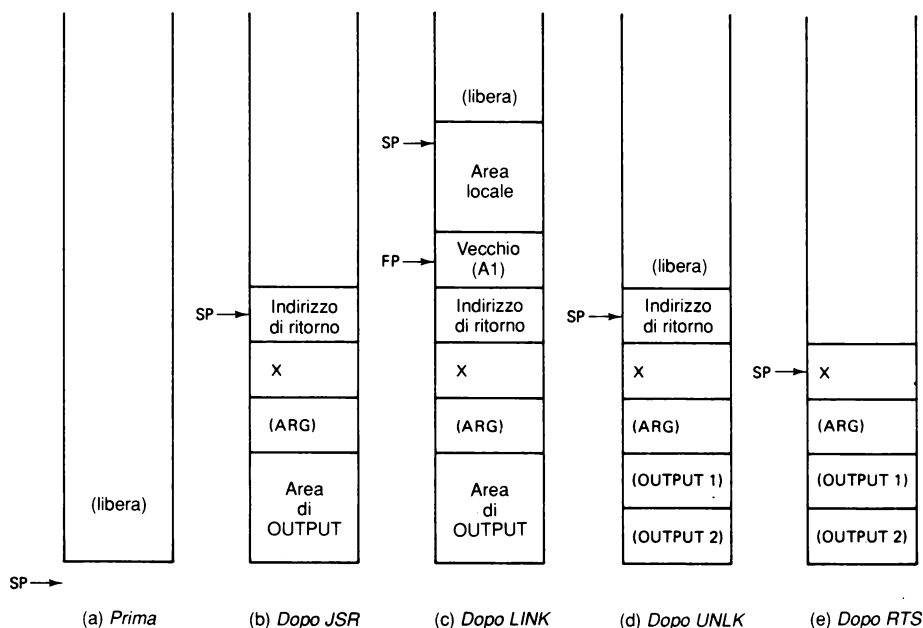



Fig. 9.18 *Contenuti di stack che utilizzano frame di stack.*

Le variabili locali sono memorizzate in quest'area ed accedute mediante spostamenti dal valore nel puntatore di frame. Una volta che gli argomenti d'ingresso sono stati elaborati e che le uscite sono state memorizzate sullo stack, viene eseguita l'istruzione UNLK. Questa istruzione, definita nella Tab. 9.14, rilascia l'area locale e ripristina il contenuto del puntatore di stack cosicché esso punterà all'indirizzo di ritorno.

In particolare, l'istruzione UNLK carica dapprima in (SP) il valore contenuto nel puntatore di frame A1, che punta al vecchio valore di A1 salvato nello stack dall'istruzione LINK. Dopodiché A1 viene ripristinato al suo valore precedente usando il modo d'indirizzamento con autoincremento, cosicché (SP) punta ora all'indirizzo di ritorno. L'istruzione RTS restituisce il controllo al programma chiamante, con (SP) che indica la locazione della cima dell'area di parametri, predisposta da questo programma come indicato nella Fig. 9.18. La routine chiamante sommerà quindi il valore 8 a (SP) mediante l'istruzione

```
ADD.L     #8,SP
```

che oltrepassa l'area di parametri d'ingresso e fa sì che (SP) punti agli argomenti di uscita. Questi argomenti rappresentano ora valori d'ingresso per il programma chiamante. Dopo che questi valori saranno stati prelevati dallo stack, il puntatore di stack riavrà il suo contenuto originale.

Per fare ritorno dalla subroutine, invece dell'istruzione RTS si potrebbe usare l'istruzione RTD (*ReTurn and Deallocate parameters:* ritorna e dealloca parametri). Oltre a restituire il controllo al programma chiamante, l'istruzione della forma

```
RTD        #<spostamento>
```

aggiunge uno spostamento di 16 bit (esteso di segno) al puntatore di stack. Il problema nell'esercizio 9.4.7 richiede l'uso dell'istruzione RTD.

ESERCIZI

9.4.1 Si confronti il passaggio di indirizzi col passaggio di valori di dati come argomenti quando una subroutine elabora un array.

9.4.2 Si discutano i vantaggi e gli svantaggi del passaggio di parametri in-linea.

9.4.3 La natura dinamica dello stack impiegato per contenere gli argomenti può risultare in un considerevole risparmio di memoria, rispetto all'assegnazione di singole aree di parametri per ciascuna subroutine. Come si determina la dimensione massima dello stack richiesta per contenere i parametri?

9.4.4 Si scriva una subroutine per confrontare due interi in precisione multipla (64 bit) e si ponga il valore massimo in una certa locazione. Si passino gli indirizzi sullo stack col primo intero nella locazione N1, il secondo in N2, mentre il risultato dovrà essere posto nella locazione MAX. Ci si accerti di correggere il valore del puntatore di stack in modo da "svuotare" lo stack prima che la subroutine restituisca il controllo al programma chiamante.

9.4.5 Si confronti la sequenza di istruzioni

```
LEA     $2000,A3
LEA     $1FF0,SP
```

con l'istruzione

```
LINK    A3, #-$10
```

se (SP) = $2000 quando viene eseguita l'istruzione LINK.

9.4.6

Si scriva un programma che produca la somma ed il valor medio di *N* interi positivi di 16 bit contenuti in un'area fissa della memoria. S'impieghi un frame di stack per il passaggio di tutti i parametri tra i segmenti di programma.

9.4.7

Si modifichi il programma dell'esempio 9.13 usando l'istruzione RTD per "aggiustare" il puntatore di stack prima del completamento della subroutine.

CAPITOLO 10

FUNZIONAMENTO DEL SISTEMA

Nei capitoli dal 6 al 9 sono stati considerati perlopiù dei programmi applicativi. Pertanto sono stati posti in rilievo l'insieme di istruzioni dell'MC68020 e le tecniche di programmazione. Nei prossimi quattro capitoli, a partire da questo, sarà discusso il funzionamento dei sistemi basati sull'MC68020. Sarà posta in evidenza l'interazione tra le varie componenti di un sistema di computer, tra cui il sistema operativo o programma supervisore, la CPU, la memoria e l'hardware relativo.

In questo capitolo sarà presentato il funzionamento generale di un sistema basato sull'MC68020, senza entrare nei dettagli, che saranno invece trattati nei prossimi capitoli. Una rappresentazione schematica di tale funzionamento generale è illustrata nella Fig. 10.1. All'accensione della macchina, viene eseguita la sequenza di reset mostrata nella Fig. 10.1(a) per inizializzare il sistema. Tale sequenza è avviata dalla circuiteria esterna, la cui linea di segnale RESET produce l'inizializzazione della CPU.[1] Successivamente l'hardware del sistema ed il programma supervisore provvedono ad inizializzare i valori in certi registri e gli indirizzi dei vettori associati col funzionamento del sistema. Questi indirizzi sono contenuti in una tabella di vettori della CPU e puntano alle locazioni iniziali delle routine di gestione delle interruzioni. Dopo l'inizializzazione, il controllo viene solitamente passato ad un programma applicativo, che opera come illustrato nella Fig. 10.1(b).

L'esecuzione del programma applicativo prosegue finché il suo compito è stato completato o finché non è richiesto un servizio di supervisore. Il programma può eseguire un'istruzione TRAP per riportare il controllo al supervisore, affinché sia svolto un determinato servizio, come l'immissione o l'emissione di dati. Nella Fig. 10.1(b), un'istruzione TRAP fa sì che il supervisore porti a termine il servizio e riporti il controllo al programma applicativo. Una volta che il programma stesso è stato completato, il controllo viene restituito al programma supervisore, che successivamente passa il controllo della CPU al successivo programma applicativo da eseguire.

---

[1] Le linee di segnale per l'MC68020 sono state presentate nel cap. 4. Le linee di segnale saranno tutte discusse in dettaglio nel cap. 13.

(a) *Inizializzazione.*

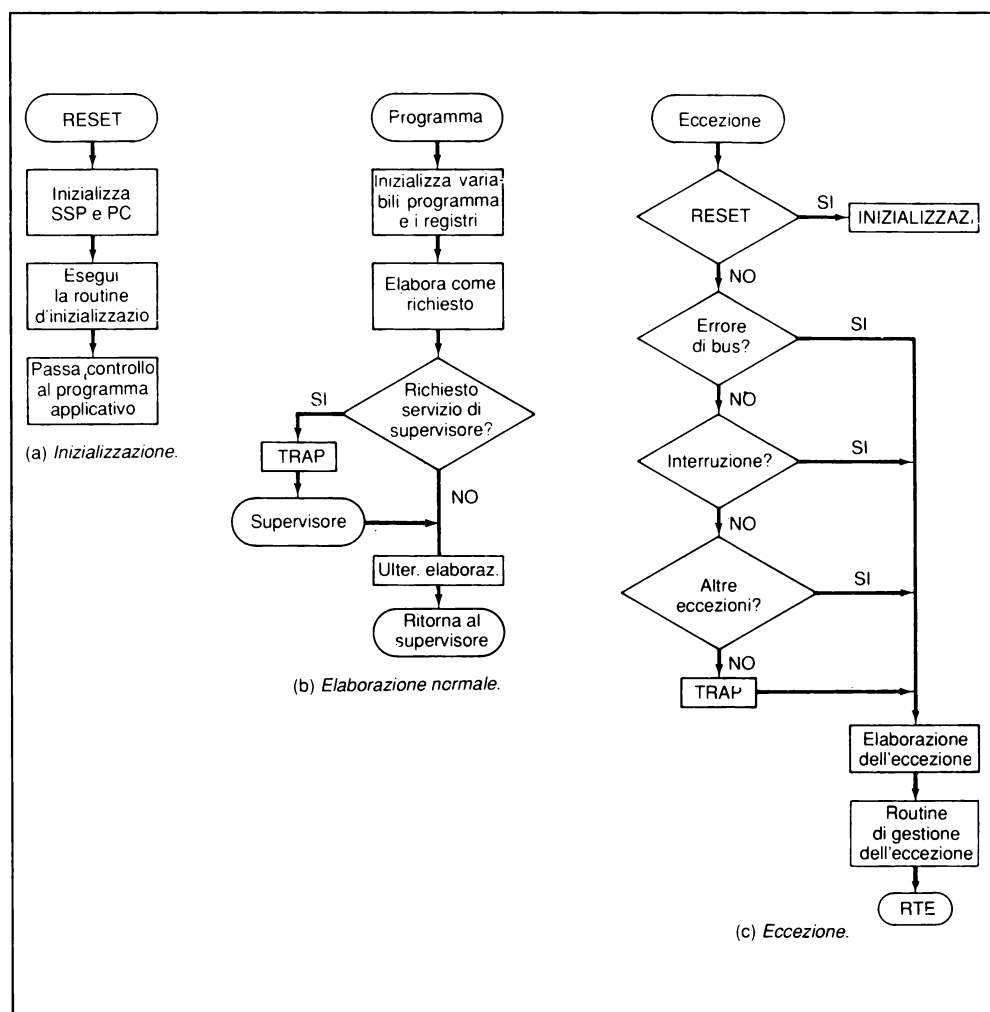
(b) *Elaborazione normale.*

(c) *Eccezione.*

Fig. 10.1 *Operazioni del sistema per un computer tipico basato sull'MC68020.*

Durante l'esecuzione di un programma applicativo, si può presentare un certo numero di eccezioni — come mostrato in Fig. 10.1(c) — che causano il passaggio del controllo ad apposite routine del supervisore per la gestione delle eccezioni. Per esempio, una richiesta di servizio da parte di un dispositivo esterno è segnalata alla CPU tramite una richiesta di errore di bus o di un'interruzione.[2] Varie altre eccezioni indicano condizioni di errore o richieste di servizio dovute al programma in corso di esecuzione. L'istruzione TRAP è un esempio di questo tipo di eccezione. Allorché viene riconosciuto un qualunque tipo di eccezione, la CPU entra

[2] Queste richieste sono avanzate tramite le linee di segnale della CPU, come sarà descritto nel par. 13.4. Tutte le eccezioni dell'MC68020 saranno discusse nel cap. 11.

dapprima nella modalità di supervisore e memorizza le informazioni appropriate nello stack di supervisore durante l'*elaborazione dell'eccezione*. Dopodiché, viene eseguita una specifica routine di gestione dell'eccezione. La sua funzione dipende interamente dal progetto della routine di supervisore progettata per gestire l'eccezione. In molti casi, come in quello dell'istruzione TRAP, la routine di gestione dell'eccezione termina e restituisce il controllo al programma che era in corso di esecuzione nel momento in cui si era presentata l'eccezione. L'istruzione RTE (*ReTurn from Exception:* ritorna dall'eccezione) ripristina le informazioni presenti nello stack e restituisce il controllo al programma applicativo.

Il par. 10.1 descrive i vari stati e le diverse modalità della CPU durante l'esecuzione del programma e la gestione delle eccezioni. Successivamete, nel par. 10.2, saranno descritte le istruzioni di controllo del sistema dell'MC68020. Tali istruzioni sono impiegate in un programma in modo supervisore per controllare la CPU ed altri elementi del sistema di computer. Infine, nel par. 10.3, sarà presentata la sequenza d'inizializzazione del sistema, oltre ad un esempio di programmazione. Il lettore dovrebbe avere già una certa familiarità con l'organizzazione di un sistema basato sull'MC68020, presentata nel cap. 2, come pure con l'impiego di registri per la CPU, comprese le differenze tra i modi di utente e di supervisore, descritti nel par. 4.2.

10.1 STATI E MODI DEL PROCESSORE

L'MC68020 opera in uno di tre stati di elaborazione: normale, di eccezione o di arresto (*halt*). Quando un programma è in esecuzione, l'attività del processore è ulteriormente descritta dal suo modo (modalità) o stato di privilegio. Durante il funzionamento normale, tale modo può essere di supervisore o di utente, in base all'impostazione del bit di stato (bit "S") nel registro di stato. Allorché si presenta un'eccezione, il processore viene posto automaticamente nella modalità di supervisore. Gli stati e i modi influiscono sia sulla programmazione che sull'attività dell'hardware del sistema. Inoltre, le linee di segnale della CPU possono essere utilizzate dalla circuiteria esterna per distinguere lo spazio d'indirizzi nella memoria della CPU in spazio di supervisore e spazio di utente. Ciò previene l'insorgere di conflitti tra programmi eseguiti in modalità differenti, come sarà spiegato nel par. 13.4.

10.1.1 Stati normale, di eccezione e di arresto

La Tab. 10.1 riassume i tre stati del processore. Lo stato *normale* è associato con l'esecuzione del programma nel modo di supervisore o in quello di utente. In questo stato, il processore preleva le istruzioni e gli operandi dalla memoria durante l'esecuzione del programma. Tale stato viene tipicamente attivato dopo l'inizializzazione del sistema. A meno che non venga rivelato un malfunzionamento dell'hardware, lo stato del processore cambia solamente quando si verifica una eccezione.

Tab. 10.1 *Stati e modi del processore.*

Stato	Condizione	Attività della CPU
Normale	Elaborazione	Esecuzione del programma nel modo di supervisore o di utente.
	Sospensione	Attesa di interruzione.
Eccezione	Reset	Inizializzazione.
	Interruzione	Riconoscimento ed elaborazione dell'interruzione.
	Trappola	Elaborazione della trappola.
	Traccia	Traccia di singola istruzione o traccia sul cambiamento di flusso.
Arresto	Condizione di errore del sistema	Nessuna attività.

Nota: Le eccezioni elencate nella Tab. 10.1 sono esempi selezionati. Un elenco completo delle eccezioni per l'MC68020 sarà fornito nel cap. 11.

L'unico caso speciale durante l'attività nello stato normale è la condizione di *sospensione* (*stop*). In questo caso il processore non continua ad eseguire le istruzioni ma resta in attesa del verificarsi di un evento esterno prima di riprendere le normali operazioni. Una condizione di sospensione nello stato normale si presenta allorché viene eseguita l'istruzione

```
STOP        #<d16>
```

Il valore immediato <d16> di 16 bit sostituisce il contenuto del registro di stato, mentre il (PC) viene fatto avanzare per puntare all'istruzione successiva. Finché non sarà stata riconosciuta un'interruzione o qualche altro evento esterno, il processore sospenderà il prelievo e l'esecuzione delle istruzioni. L'istruzione STOP dev'essere eseguita da un programma nel modo di supervisore, altrimenti ne risulterebbe una violazione di privilegio (trappola).[3] L'attività del processore viene ripristinata allorché viene riconosciuta un'interruzione o quando viene eseguito un reset del sistema. In pratica, l'istruzione STOP può essere considerata come un'istruzione di "attesa d'interruzione", utilizzata soltanto in occasioni speciali.

[3] Si presenterà un'eccezione di traccia se viene indicata la condizione di traccia allorché l'istruzione STOP viene eseguita. L'eccezione di violazione di privilegio sarà descritta nel par. 11.4.

La CPU entra nello stato di *eccezione* allorché viene riconosciuto un reset, un'interruzione, una trappola, una tracca o qualche altra eccezione. Il processore viene posto automaticamente nel modo di supervisore, dopodiché inizia l'elaborazione dell'eccezione. La Fig. 10.2 illustra questa sequenza di eccezione in due parti. In base alla descrizione fornita dalla Motorola per l'MC68020, l'*elaborazione dell'eccezione* per la maggior parte delle eccezioni include la modifica del modo della CPU in quello di supervisore, la determinazione del numero di vettore relativo all'eccezione nella tabella di vettori della CPU, ed il salvataggio nello stack di supervisore del registro di stato (SR) della CPU, del contatore di programma (PC) e di altre informazioni.[4] Dopodiché, l'indirizzo del vettore che punta alla routine di gestione dell'eccezione viene caricato automaticamente nel PC ed il controllo viene passato alla routine. La CPU inizia quindi l'elaborazione normale finché la routine non viene completata. Se la routine esegue infine l'istruzione RTE (*ReTurn from Exception:* ritorna dall'eccezione) allora la CPU riprenderà ad eseguire il programma che era stato interrotto nel momento in cui si era presentata l'eccezione. La normale esecuzione del programma viene ripresa allorché la CPU ripristina negli appropriati registri del processore i valori salvati di (SR) e (PC) e le altre informazioni memorizzate nello stack di supervisore.[5]

Flusso del programma
Elaborazione normale
Elaborazione dell'eccez.
Elaborazione normale
Routine di gestione dell'eccez.
RTE

Nota: L'elaborazione dell'eccezione viene eseguita automaticamente dalla CPU allorché viene riconosciuta un'eccezione. Il controllo è quindi passato alla routine di gestione dell'eccezione associata con la particolare eccezione che si è verificata.

Fig. 10.2 *Sequenza di eccezione.*

[4] Nessuna informazione viene salvata per l'eccezione di reset. La tabella di vettori contiene gli indirizzi di routine di gestione delle eccezioni, come descritto nel par. 10.3.

[5] Per certi tipi di errore, il programma supervisore potrebbe non restituire il controllo al programma che era in corso di esecuzione nel momento in cui si è presentata l'eccezione. Il risultato dipende interamente dal progetto della routine di supervisore che gestisce l'eccezione.

Lo stato di *arresto* (*halt*) provvede alla protezione del sistema, facendo sì che la CPU cessi qualsiasi attività di segnalazione esterna. Come sarà spiegato nel cap. 11, il processore si arresta se vengono rivelati certi tipi di errore mentre esso sta già elaborando un altro errore. Queste condizioni di errore dovrebbero presentarsi soltanto in seguito ad un difetto così grave dell'hardware da rendere impossibile qualsiasi rimedio, per cui è necessario un reset del sistema per riavviare il processore arrestato. Durante lo stato di arresto, la CPU indica la propria condizione sul bus di sistema, tramite una linea di segnale predisposta esplicitamente per tale scopo. Spetterà poi alla circuiteria esterna o all'operatore la decisione di riavviare il sistema arrestato.

Un processore nella condizione di sospensione o in quella di arresto non può essere riavviato da un programma, poiché l'attività della CPU viene controllata da linee di segnale esterne dopo che si è verificata una di queste due condizioni. Per contro, durante l'elaborazione normale o di eccezione, è un programma ad avere il controllo del processore. La distinzione tra programmi operanti nel modo di supervisore e programmi operanti nel modo di utente si rivela importante quando il processore opera nello stato normale. Durante le operazioni normali, il modo definisce le condizioni di privilegio per un programma.

10.1.2 I modi di supervisore e di utente

Alcune differenze tra il modo di supervisore e quello di utente dell'MC68020 sono già state presentate nei capitoli precedenti. L'impiego di un puntatore di stack separato per ciascuna modalità ed altre importanti distinzioni sono elencate nella Tab. 10.2. Essenzialmente, il modo di supervisore rappresenta il livello con più privilegio. Un programma operante in questa modalità può eseguire qualsiasi istruzione dell'MC68020 e può modificare il registro di stato ed altri registri speciali. Per i programmi nel modo di utente è ammesso un insieme ristretto di istruzioni senza alcun privilegio per eseguire certe istruzioni che controllano l'attività del sistema.

Come indicato nella Tab. 10.2, il modo di supervisore viene attivato allorché un'eccezione di qualsiasi tipo viene riconosciuta dalla CPU. Un programma operante nel modo di supervisore può commutare nel modo di utente modificando il bit di stato (SR)[13] nel registro di stato. Ogniqualvolta (SR)[13] = {1}, il processore sta operando nel modo di supervisore; la transizione al modo di utente è ottenibile ponendo (SR)[13] = {0}. I metodi per effettuare tale transizione ed altre distinzioni tra il modo di supervisore e il modo di utente saranno l'argomento di questo e dei prossimi sottoparagrafi.

Il modello di programmazione. I registri generali D0-D7, A0-A6, il contatore di programma (PC) ed il registro dei codici di condizione (CCR) sono disponibili a programmi sia nel modo di supervisore che in quello di utente. Il salvataggio del contenuto di qualsiasi registro generale sullo stack di sistema è affidato alla responsabilità del programmatore quando il modo operativo della CPU viene cambiato ed il controllo è passato ad un nuovo programma. Il programma dovrà anche

Tab. 10.2 *Distinzioni tra i modi di supervisore e di utente.*

	Modo di supervisore	**Modo di utente**
Modo inserito da:	Riconoscimento di trappola, reset o interruzione	Azzeramento del bit di stato "S"
Puntatore di stack di sistema	Puntatore di stack di supervisore	Puntatore di stack di utente (A7)
Altri puntatori di stack	Puntatore di stack di utente (USP) e registri A0-A6	Registri A0-A6
Bit di stato disponibili		
Lettura	C, V, Z, N, X, I_0-I_2, S, T0, T1	C, V, Z, N, X
Scrittura	C, V, Z, N, X, I_0-I_2, S, T0, T1	C, V, Z, N, X
Istruzioni disponibili	Tutte, incluse le istruzioni di controllo del sistema	Tutte, tranne quelle elencate per il modo di supervisore
Registri speciali	CAAR, CACR, DFC, SFC, VBR	---

Note:
1. I registri d'indirizzo A0-A6 possono essere utilizzati come puntatori di stack privati da programmi eseguibili in entrambe le modalità.
2. I registri speciali sono quelli che controllano la memoria cache (CAAR, CACR), le linee di segnale dei codici di funzione (DFC, SFC) e la locazione della tabella di vettori (VBR). Le spiegazioni di questi registri sono fornite nei paragrafi appropriati del testo.

ripristinare i valori in tali registri prima che il modo sia riportato a quello precedente, qualora il controllo debba essere restituito al programma interrotto.

Sia nel modo di supervisore che in quello di utente, la designazione A7 o SP del linguaggio assembler si riferisce allo stack di sistema attivo. Invece, la designazione USP è il nome mnemonico (*User Stack Pointer:* puntatore di stack di utente) impiegato in un programma di modo supervisore che deve leggere o modificare il contenuto del puntatore di stack di utente. Questa è la maniera in cui un programmatore in linguaggio assembler, che scrive un programma da eseguire nel modo di supervisore, può effettuare la distinzione tra il puntatore di stack di supervisore (A7 o SP) ed il puntatore dello stack di sistema nel modo di utente.

La Fig. 10.3(a) illustra il modello generale di programmazione per l'MC68020. Il riferimento ad uno qualunque dei registri indicati può avvenire in entrambe le modalità tramite certe istruzioni del linguaggio assembler, che sono state descritte ed

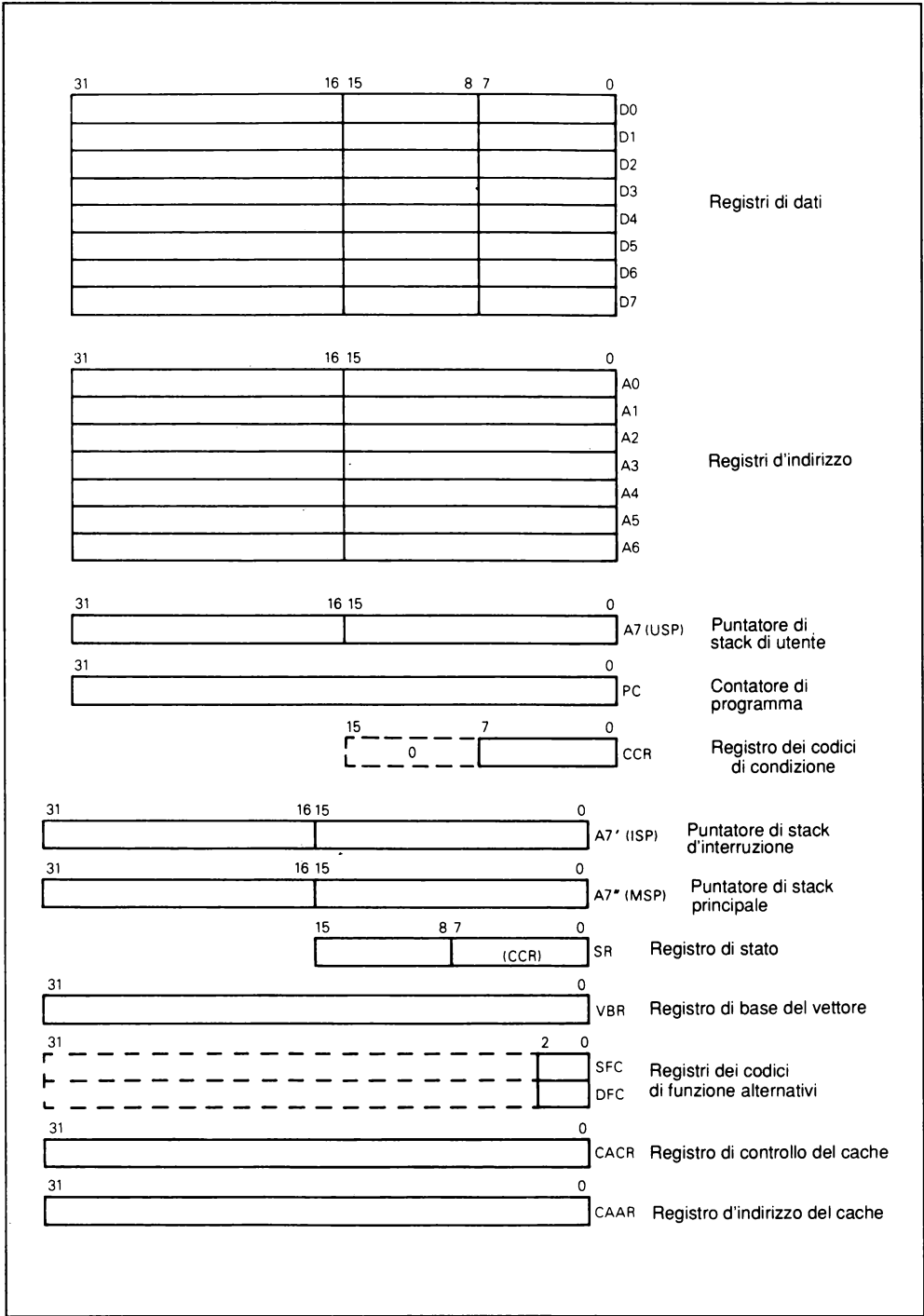



Fig. 10.3 *(a) Modello di programmazione generale per l'MC68020. (b) Supplemento di modello di programmazione di supervisore per l'MC68020. (Per gentile concessione di Motorola, Inc.)*

utilizzate nei capitoli dal 4 al 9. I registri addizionali disponibili ad un programma nel modo di supervisore sono mostrati nella Fig. 10.3(b). Lo scopo e la programmazione per alcuni di questi registri saranno descritti nel par. 10.2. Gli altri registri di supervisore saranno descritti nei prossimi capitoli.

ESERCIZI

10.1.1 Si tracci un diagramma che mostri i possibili stati dell'MC68020 e le transizioni tra di essi.

10.1.2 Si confronti la condizione di sospensione con lo stato di arresto.

10.1.3 Si discutano le possibili applicazioni dell'istruzione STOP e la condizione di sospensione del processore.

10.1.4 Si elenchino i vari metodi di protezione disponibili per un sistema basato sull'MC68020 e si definiscano lo scopo e le possibili applicazioni di ciascun metodo. Si includano considerazioni sia sull'hardware che sul software.

10.2 ISTRUZIONI DI CONTROLLO DEL SISTEMA

L'MC68020 ha un certo gruppo di istruzioni adibite al *controllo del sistema.* Si tratta di istruzioni privilegiate, che sono utilizzate per modificare dinamicamente l'attività del sistema del computer. La Tab. 10.3 elenca le istruzioni ed i relativi operandi. Un certo numero di esse servono a modificare il contenuto del registro di stato (SR). Pertanto, queste istruzioni sono considerate *privilegiate* perché possono "manipolare" il contenuto di SR. L'istruzione MOVE può essere usata per modificare il contenuto del puntatore di stack di utente (USP). Il registro dei codici di condizione (CCR) rappresenta una valida sorgente o destinazione per programmi operanti nel modo di supervisore o in quello di utente. Anche se non è designato come operando nell Tab. 10.3, il contenuto del CCR può essere manipolato da qualsiasi istruzione che consenta di designare come operando il contenuto del registro di stato.

L'istruzione RTE (*ReTurn from Exception:* ritorna da eccezione) è impiegata per trasferire il controllo da una routine di gestione dell'eccezione ad un altro programma. Un'istruzione RESET viene solitamente inviata da una routine d'inizializzazione per far sì che vari chip periferici assumano il loro stato iniziale prima di essere utilizzati per le operazioni d'ingresso/uscita. L'istruzione MOVEC (*MOVE Control register:* trasferisce registro di controllo) serve a manipolare i contenuti di registri speciali che fanno parte dell'insieme di registri di supervisore, come definito nel precedente par. 10.1.

Tab. 10.3 *Istruzioni di controllo del sistema.*

ISTRUZIONE	SINTASSI DI OPERANDO	DIMENSIONE DI OPERANDO	OPERAZIONE
			Privilegiata
ANDI	#<dato>,SR	16	Dato immediato $\wedge$ SR $\rightarrow$ SR
EORI	#<dato>,SR	16	Dato immediato $\oplus$ SR $\rightarrow$ SR
MOVE	<EA>,SR SR,<EA>	16 16	Sorgente $\rightarrow$ SR SR $\rightarrow$ destinazione
MOVE	USP,An An,USP	32 32	USP $\rightarrow$ An An $\rightarrow$ USP
MOVEC	Rc,Rn Rn,Rc	32 32	Rc $\rightarrow$ Rn Rn $\rightarrow$ Rc
MOVES	Rn,<EA> <EA>,Rn	8, 16, 32	Rn $\rightarrow$ destinazione tramite DFC Sorgente tramite SFC $\rightarrow$ Rn
ORI	#<dato>,SR	16	Dato immediato $\vee$ SR $\rightarrow$ SR
RESET	-----	---	Attiva la linea $\overline{\text{RESET}}$
RTE	-----	---	((SP)) $\rightarrow$ SR; (SP) + 2 $\rightarrow$ (SP); (SP) $\rightarrow$ PC; (SP) + 6 $\rightarrow$ (SP); ripristina lo stack in base al suo formato
STOP	#<dato>	16	Dato immediato $\rightarrow$ SR; STOP

Note:

1. $\wedge$ = AND logico.
2. $\vee$ = OR logico.
3. $\oplus$ = OR esclusivo.
4. Rc è un qualsiasi registro generale di controllo o il puntatore di stack di utente (CAAR, CACR, DFC, ISP, MSP, SFC, USP, VBR).
5. Rn è un qualsiasi registro d'indirizzo o registro di dati.

Il presente paragrafo 10.2 descrive la maggior parte delle istruzioni elencate nella Tab. 10.3, per definire la loro sintassi in linguaggio assembler e le operazioni funzionali. L'istruzione STOP è stata descritta in precedenza nel par. 10.1. L'istruzione MOVES (*MOVE address Space:* trasferisce spazio d'indirizzi) sarà descritta nel par. 10.4.

10.2.1 Modifica del registro di stato

La Tab. 10.4 elenca le istruzioni disponibili ad un programma in modo supervisore che può modificare il contenuto del registro di stato. Le istruzioni logiche (ANDI, EORI, ORI) operano nella maniera descritta nel cap. 8. Tuttavia, in questa applicazione, la locazione di destinazione è il registro di stato, che le rende istruzioni privilegiate. Ognuna di queste istruzioni svolge l'operazione logica designata usando il valore immediato di 16 bit e l'intero contenuto del registro di stato.

L'istruzione MOVE è usata per trasferire l'operando di sorgente al registro di stato, sostituendo così il contenuto precedente. Nella forma:

```
MOVE      <EA>,SR
```

l'indirizzo effettivo <EA> è specificato da una qualsiasi modalità d'indirizzamento di dati, che comprende tutte le modalità d'indirizzamento, tranne quella diretta di registro d'indirizzo. Per trasferire il contenuto del registro di stato ad una locazione di memoria o ad un registro di dati, s'impiega l'istruzione

```
MOVE.W    SR,<EA>
```

Tab. 10.4 *Istruzioni per modificare lo stato del processore.*

Sintassi		Operazione
ANDI.W	#<d_{16}>,SR	(SR) ← (SR) AND <d_{16}>
EORI.W	#<d_{16}>,SR	(SR) ← (SR) EOR <d_{16}>
MOVE.W	<EA>,SR	(SR) ← (EA)
ORI.W	#<d_{16}>,SR	(SR) ← (SR) OR <d_{16}>

Note:
1. Tutte le istruzioni sono privilegiate.
2. Una MOVE a SR richiede un modo d'indirizzamento di dati per <EA>. Quindi sono ammessi tutti i modi d'indirizzamento, tranne quello diretto di registro d'indirizzo, per l'operando di sorgente.
3. Il contenuto di SR viene letto con l'istruzione MOVE.W SR,<EA>. La destinazione può essere definita da qualsiasi modo alterabile di dati; sono proibiti l'indirizzamento diretto di registro d'indirizzo e quello relativo al PC.

Esempio 10-1

La Fig. 10.4 mostra il registro di stato dell'M68020, suddiviso in byte di utente SR[7:0] e in byte di sistema SR[15:8]. I valori nella colonna di AND e OR sono forniti in esadecimale. La maschera AND azzera qulunque bit nel registro di stato che corrisponde ad uno {0} nella maschera. L'OR di (SR) con la configurazione di bit indicata come {ABILITAZ.} pone a {1} il bit corrispondente. Quindi l'istruzione

```
ANDI    #$7FFF,SR
```

impone T1 = {0} per disabilitare il modo di traccia di singola istruzione, ma non interessa gli altri bit. L'istruzione

```
ORI     #$8000,SR
```

abilita il modo di traccia di singola istruzione. L'impiego degli altri bit è analogo. Per esempio, quando il bit 13 vale {0}, il processore sta operando nel modo di utente.

Il valore del livello d'interruzione {I_2, I_1, I_0} non è trattato come una variabile logica, ma come un intero di 3 bit. Un valore di 0, {000} indica che tutti i livelli d'interruzione saranno accettati con priorità crescente da 1 a 7. Durante l'elaborazione dell'interruzione, il valore indica il livello di corrente, come spiegato nel cap. 11. Le interruzioni a questo livello e a livelli inferiori vengono ignorate. L'istruzione

```
ANDI    #$8FF,SR
```

abilita tutti i livelli d'interruzione, ponendo a {000} tutti i bit d'interruzione. I livelli d'interruzione sono disabilitati dall'istruzione

```
ORI     #$0700,SR
```

che disabilita tutti i livelli inferiori al settimo, poiché il livello 7 non può essere mascherato (disabilitato).

Nella Fig. 10.4 sono mostrate anche le operazioni sul byte di utente o su CCR. I valori immediati elencati non interessano il byte di sistema quando viene usato con le istruzioni ANDI o ORI come mostrato. Per esempio, l'istruzione

```
ANDI    #$FF00,SR
```

causerebbe l'azzeramento del CCR. Quindi l'impiego di ANDI col valore di {MASCHERA} indicato assegnerebbe il valore {0} al bit corrispondente nel CCR. Il bit viene posto a {1} quando ORI viene usata col valore immediato specificato come {ABILITAZ.} nella Fig. 10.4.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0 ← Numero di
T1	T0	S	M	0	I_2	I_1	I_0	0	0	0	X	N	Z	V	C
← Byte di sistema →								← Byte di utente →							

Condizione	Bit di stato	AND {MASC.}	OR {ABILITAZ.}
Modo di traccia (singola)	(SR) [15] = T1	{7FFF}	{8000}
Modo di traccia (cambio)	(SR) [14] = T0	{BFFF}	{4000}
Modo di supervisore	(SR) [13] = S	{DFFF}	{2000}
Stack principale	(SR)[12] = M	{EFFF}	{1000}
Livello d'interruzione	(SR) [10:8] = livello	{F8FF}	{0700}
Estensione	(SR) [4] = X	{FFEF}	{0010}
Negativo	(SR) [3] = N	{FFF7}	{0008}
Zero	(SR) [2] = Z	{FFFB}	{0004}
Overflow	(SR) [1] = V	{FFFD}	{0002}
Riporto	(SR) [0] = C	{FFFE}	{0001}

Note:

(1) SR[15:8] è il byte di sistema; SR[7:0] è il byte di utente o CCR.

(2) ((SR) AND {MASCHERA}) pone il bit a {0}; ((SR) OR {ABILITAZ.}) pone il bit a {1}.

(3) Per il livello d'interruzione, il valore $\{I_2I_1I_0\}$ è interpretato come un codice di 3 bit.

(4) Può essere abilitata sia la traccia sulla singola istruzione (T1) che la traccia sul cambio del controllo di flusso (T0). L'abilitazione simultanea di entrambe risulta in un'operazione indefinita, come descritto nell'*MC68020 User's Manual* della Motorola.

(5) Quando M = {0}, il puntatore di stack d'interruzione (ISP) è il puntatore di stack di supervisore attivo.

Fig. 10.4 *Operazioni del registro di stato.*

10.2.2 Manipolazione del puntatore di stack di utente

Un programma operante nel modo di supervisore può salvare, ripristinare o modificare il contenuto del puntatore di stack di utente. L'istruzione privilegiata

```
MOVE.L      USP,<An>
```

copia (USP) nel registro d'indirizzo An. Il trasferimento opposto ha la forma

```
MOVE.L      <An>,USP
```

ed è usato per inizializzare o modificare (USP). In ciascun caso, ha luogo un trasferimento di 32 bit.

Come ci si potrebbe aspettare, un programma operante nel modo di supervisore ha il controllo del contenuto iniziale del puntatore di stack di utente. Pertanto, l'indirizzo dello stack di utente dev'essere caricato nell'USP dal programma supervisore durante l'inizializzazione. L'indirizzo appropriato potrebbe essere trasferito in <An> e poi al puntatore di stack di utente mediante le istruzioni

```
MOVEA.L     #USERSTK,<An>
MOVE.L      <An>,USP
```

dove USERSTK è l'indirizzo del fondo dello stack di utente.[6] Un programma nel modo di utente fa riferimento al puntatore di stack di utente specificando A7 o SP in un'istruzione del linguaggio assembler, come già discusso nel par. 10.1.

10.2.3 Manipolazione del registro dei codici di condizione

Le istruzioni elencate nella Tab. 10.5 sono disponibili sia ai programmi in modo supervisore che a quelli in modo utente. Le istruzioni logiche permettono di modificare il contenuto del CCR usando un valore immediato di 8 bit. Per esempio, l'istruzione

```
ORI.B       #$01,CCR
```

assegna il valore {1} al bit di riporto e non modifica alcun bit nel registro dei codici di condizione. L'intero contenuto del CCR può essere modificato dall'istruzione

```
MOVE.W      <EA>,CCR
```

in cui (<EA>)[7:0] contiene i nuovi bit dei codici di condizione per il CCR. L'indirizzamento per <EA> richiede un modo d'indirizzamento di dati che consente tutte le

[6] Un programma in modo di supervisore può usare l'istruzione MOVEC per agire sul contenuto dell'USP. Questa istruzione sarà descritta nel par. 10.2.6.

Tab. 10.5 *Istruzioni per modificare (CCR).*

Sintassi		**Operazione**
Modifica del CCR		
ANDI.B	#<d8>,CCR	(SR)[7:0] ← (SR)[7:0] AND <d8>
EORI.B	#<d8>,CCR	(SR)[7:0] ← (SR)[7:0] EOR <d8>
MOVE.W	#<EAs>,CCR	(SR)[7:0] ← (EAs)[7:0]
ORI.B	#<d8>,CCR	(SR)[7:0] ← (SR)[7:0] OR <d8>
MOVE.W	CCR,<EA>	(EA) ← (CCR)

Note:
1. Una MOVE a CCR richiede un modo d'indirizzamento di dati per <EAs>; l'indirizzo di sorgente non può essere specificato dal modo diretto di registro d'indirizzo.
2. CCR è (SR)[7:0].
3. La destinazione nell'istruzione MOVE.W CCR,<EA> dev'essere indirizzata da un modo d'indirizzamento alterabile di dati; sono proibiti il modo diretto di reglistro d'indirizzo e quello relativo al PC.

modalità tranne il modo diretto di registro d'indirizzo. Si noti che l'operazione richiede un operando di word ma soltanto il byte meno significativo è usato per aggiornare i codici di condizione.

10.2.4 L'istruzione RTE

L'istruzione di ritorno dall'eccezione (*ReTurn form Exception:* RTE) è un'istruzione privilegiata usata per caricare il registro di stato ed il contatore di programma con valori memorizzati nello stack di supervisore. L'operazione dell'istruzione RTE per l'MC68020 è:

(a) Caricamento di (SR)

```
(SR)[W] ← ((SSP))          ; PRELIEVO DI (SR)
(SSP) ← (SSP) + 2          ; PUNTAMENTO AL (PC)
```

(b) Caricamento di (PC)

```
(PC)[L] ← ((SSP))          ; PRELIEVO DI (PC)
(SSP) ← (SSP) + 6
```

(c) Ripristino dei valori interni dei registri con informazioni prelevate dallo stack.

Fig. 10.5
Operazione dell'istruzione RTE.

(libera)
↑ Memoria bassa
SP → "SR"
"PC" [31:16]
"PC" [15:0]
Formato

(a) *Stack di sistema prima di RTE.*

(libera)
SP →

(b) *Dopo RTE.*

(SR) = "SR"
(PC) = "PC"

Nota: "SR" e "PC" sono i valori da sostituire a quelli correnti di SR e PC, rispettivamente.

Ciò è illustrato nella Fig. 10.5. L'istruzione RTE è quella d'impiego più frequente come ultima istruzione nelle routine di gestione delle eccezioni. Essa ripristina i valori di (PC) e di (SR) posti sullo stack di supervisore quando avviene un'eccezione. L'istruzione RTE è anche usata per passare il controllo ad un programma in modo di utente durante la procedura d'inizializzazione del sistema, come discusso nel prossimo paragrafo.

Lo stack mostrato nella Fig. 10.5 rappresenta il più semplice stack di quattro word creato da un'eccezione. Dopo il riconoscimento di un'eccezione, la CPU salva 4, 6, 10, 16 o 46 word sullo stack di sistema. La word di formato di 16 bit salvata con (SR) e (PC) indica la quantità d'informazione da ripristinare quando viene eseguita l'istruzione RTE. Questa word di formato ed altre informazioni contenute nello stack sono ripristinate nei registri interni della CPU che non fanno parte dell'insieme di registri programmabili della CPU. Il formato specifico dello stack per ciascuna eccezione sarà descritto nel cap. 11.

10.2.5 L'istruzione RESET

RESET è un'istruzione privilegiata utilizzata per ripristinare le condizioni iniziali delle interfacce esterne e dei dispositivi durante l'inizializzazione del sistema.

La sua esecuzione attiva una linea di segnale che serve ad indicare alla circuiteria esterna che il processore sta richiedendo l'inizializzazione delle interfacce appropriate. La funzione esatta dell'istruzione RESET in termini dell'attività del sistema è determinata dal progetto dell'hardware del sistema. La maggior parte dei chip periferici della famiglia dell'MC68020 risponde a questa istruzione (tramite una linea di segnale di RESET) inizializzando la propria circuiteria interna. Questa istruzione sarà ulteriormente descritta nel par. 13.2.

10.2.6 L'istruzione MOVEC ed i registri speciali del processore

L'MC68020 ha un certo numero di registri speciali che fanno parte del modello di programmazione di supervisore. Questi registri sono stati mostrati nella Fig. 10.3(b) del par. 10.1. Il contenuto dei registri può essere modificato dall'istruzione privilegiata MOVEC (*MOVE Control register:* trasferisci registro di controllo), definita nella Tab. 10.6. Come appare dalla tabella, l'istruzione MOVEC può trasferire valori tra i registri d'indirizzo o di dati ed i registri speciali. Questa istruzione può essere usata anche per modificare il contenuto del puntatore di stack di utente USP.

I registri denotati come speciali o i registri di controllo dell'MC68020 sono elencati nella Tab. 10.7. Le operazioni con questi registri sono descritti nei paragrafi appropriati di questo libro. I registri di controllo del cache saranno trattati nel par. 12.1. Le linee di codice di funzione saranno descritte nel par. 13.4. I puntatori di stack di supervisore saranno discussi nei parr. 11.6 e 12.5. Un esempio che illustra l'inizializzazione dell'USO è fornito nel par. 10.3. Il registro di base del vettore è usato per rilocare la tabella di vettori della CPU, come sarà descritto nel par. 10.3.

Tab. 10.6 *L'istruzione MOVEC di trasferimento del registro di controllo.*

Sintassi		Operazione
MOVEC	<Rc>,<Rn>	(<Rn>) ← (<Rc>)
MOVEC	<Rn>,<Rc>	(<Rc>) ← (<Rn>)

Note:
1. <Rn> è uno qualunque dei registri <An> o <Dn>.
2. <Rc> è uno dei registri speciali del processore o l'USP.
3. Tutte le operazioni sono su 32 bit.

Tab. 10.7 *Registri speciali del processore.*

Tipo e codice mnemonico	Definizione
Controllo del cache	
CAAR	Registro di indirizzo del cache (*CAche Address Register*)
CACR	Registro di controllo del cache (*CAche Control Register*)
Linee di codice di funzione	
DFC	Codice di funzione di destinazione (*Destination Function Code*)
SFC	Codice di funzione di sorgente (*Source Function Code*)
Puntatori di stack di supervisore	
ISP	Puntatore di stack d'interruzione (*Interrupt Stack Pointer*)
MSP	Puntatore di stack principale (*Master Stack Pointer*)
Puntatore di stack di utente	
USP	Puntatore di stack di utente (*User Stack Pointer*)
Registro di base del vettore	
VBR	Registro di base del vettor (*Vector Base Register*)

ESERCIZI

10.2.1

Si determini l'effetto delle seguenti istruzioni:

```
(a) MOVE.W   #$0400,SR
(b) ANDI.W   #$DFDD,SR
(c) MOVE.W   #$2700,SR
(d) EORI.W   #$2000,SR
```

Le istruzioni sono eseguite da un programma nel modo di supervisore.

10.2.2

Qual è l'effetto delle seguenti istruzioni eseguite da un programma nel modo di utente?

(a) MOVE.W #$000C,CCR
(b) ANDI.B #$01,CCR
(c) EORI.W #$2700,SR

10.2.3

Si scriva la sequenza di istruzioni per inizializzare (USP) al valore esadecimale $3830. Questo programma dev'essere eseguito nel modo di supervisore.

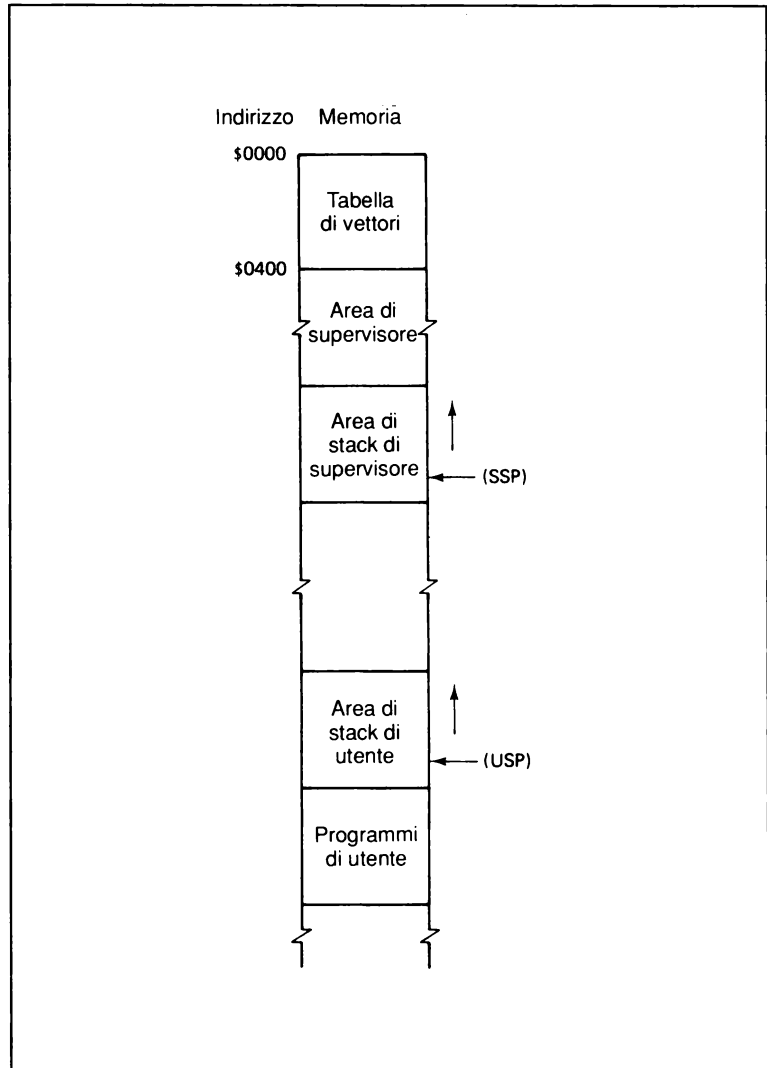
10.3 INIZIALIZZAZIONE DEL SISTEMA

La sequenza di *inizializzazione del sistema* ha lo scopo di porre il sistema in uno stato o in una condizione nota prima dell'esecuzione di qualsiasi programma applicativo. Tale sequenza inizia normalmente con un segnale di reset fornito dalla circuiteria esterna per far sì che l'hardware assuma il suo stato iniziale. Poi viene eseguita una routine d'inizializzazione, che assegna i valori iniziali alle costanti, agli indirizzi e ad altri valori di dati associati con l'attività del sistema. La routine di inizializzazione è di solito contenuta in una memoria a sola lettura e può essere eseguita automaticamente dopo un reset del sistema. Questa routine non solo assegna i valori iniziali ai dati usati per il sistema, ma può anche far sì che il sistema operativo venga caricato da un dispositivo esterno di memorizzazione, quale un'unità a disco. Naturalmente, il procedimento esatto dipende dalla configurazione dell'hardware e dall'applicazione del sistema. In questo paragrafo è descritta la procedura d'inizializzazione fondamentale per sistemi basati sull'MC68020, conformemente ai requisiti del processore.

L'inizializzazione può produrre la configurazione tipica della memoria come mostrato nella Fig. 10.6. La tabella di vettori nella maggior parte dei sistemi dell'MC68020 è situata nell'area più bassa della memoria e contiene gli indirizzi delle routine di eccezione. L'SSP e l'USP puntano al fondo dell'area di stack di supervisore e dell'area di stack di utente, rispettivamente. La quantità di spazio riservata per il programma e per le aree di stack è determinata nella fase di progettazione del sistema di software. Dev'essere allocato uno spazio sufficiente affinché le varie aree non si sovrappongano durante le operazioni. Gli stack mostrati nella Fig. 10.6 sono di sistema e crescono verso locazioni inferiori nella memoria quando indirizzi di ritorno, contenuti di registri, ed altri dati sono inseriti nello stack.

Una volta che il sistema è stato inizializzato e che i vari programmi da eseguire sono stati caricati nella memoria, il programma supervisore passa il controllo ad un programma applicativo. In questo paragrafo sono anche trattati i metodi per il trasferimento del controllo.

Fig. 10.6
Configurazione iniziale della memoria.

Indirizzo Memoria
$0000
Tabella di vettori
$0400
Area di supervisore
Area di stack di supervisore
(SSP)
Area di stack di utente
(USP)
Programmi di utente

10.3.1 La procedura d'inizializzazione

Per semplificare la discussione, la procedura d'inizializzazione è separata in due fasi. Nella prima fase, l'hardware della CPU MC68020 subisce un reset e causa l'esecuzione della routine alla locazione indicata dal valore iniziale del contatore di programma. Questa parte dell'inizializzazione è fissata dal progetto del processore e non può essere modificata. La seconda fase inizia quando viene eseguita la routine d'inizializzazione del programma supervisore.

Reset ed inizializzazione del processore. L'inizializzazione viene avviata da un dispositivo esterno che attiva la linea di segnale $\overline{\text{RESET}}$ diretta alla CPU. La sequenza degli eventi è mostrata nella Fig. 10.7, in cui il processore definisce dapprima il contenuto del registro di stato per disabilitare il tracciamento, entra nella modalità di supervisore, abilita lo stack d'interruzione, e maschera tutte le interruzioni fino al livello 7. Anche i registri VBR e CACR sono inizializzati a $00000000. Successivamente, in sequenza, la CPU fa sì che il puntatore di stack di supervisore ed il contatore di programma siano inizializzati dai primi otto byte nella memoria. Se non si presenta alcun errore, l'esecuzione del programma inizia dalla locazione definita dal contenuto del contatore di programma.

Nella maggior parte dei sistemi, le locazioni inferiori di memoria che contengono gli indirizzi dei vettori sono posti solitamente nella memoria volatile o RAM.

	Commenti
$\overline{\text{RESET}}$	Linea di segnale attivata
SR[15, 14, 12] = {0, 0, 0} SR[13] = {1} SR[10:8] = {111} (VBR) = $00000000 (CACR) = $00000000	Traccia disattivata; Modo di supervisore: Stack princip. disabilitato; Interruzioni mascherate; Vettore di base a zero; Cache disabilitato.
(SSP) ← (00-03)	Puntatore di stack iniziale
(PC) ← (04-07)	Contatore di programma iniziale
Esegui la routine d'inizializzazione che inizia da (PC)	Inizia la routine d'inizializzazione

Note:

(1) Se si verifica un errore durante il prelievo dei valori di (SSP) e di (PC), il processoree ntrerà nello stato di arresto (alt).

(2) $\overline{\text{RESET}}$ è una linea di segnale entrante nella CPU.

Fig. 10.7
Operazione di RESET.

Pertanto, se l'alimentazione elettrica venisse a mancare, si dovrebbe ritenere che tali locazioni contengano dati ed istruzioni non ammesse. Durante un reset del sistema, il processore MC68020 legge la locazione di longword (00-03) per reperire il valore del puntatore di stack di supervisore e la locazione (04-07) per il valore del contatore di programma. Questi otto byte devono essere interpretati come valori permanenti. Pertanto, dev'essere progettata un'apposita circuiteria esterna a tal fine.[7] La routine d'inizializzazione che viene successivamente eseguita può inizializzare i restanti valori nella altre locazioni di vettori e svolgere tutta l'elaborazione necessaria. Nei sistemi basati sull'MC68020, la tabella di vettori può essere rilocata nella memoria.

La tabella dei vettori di eccezione. Anche se le eccezioni specifiche consentite dall'MC68020 non saranno discusse in dettaglio fino al cap. 11, la tabella di vettori mostrata nella Tab. 10.8 elenca tutte le locazioni dei vettori di eccezione per la CPU. La tabella nella memoria contiene 256 indirizzi di 32 bit, ciascuno dei quali punta alla locazione della corrispondente routine di eccezione di I/O relativa all'eccezione. Soltanto le prime due, che forniscono gli indirizzi di reset, sono obbligatorie; le altre locazioni sono riempite da una routine d'inizializzazione nel rispetto dei requisiti definiti dal progetto dell'hardware e del software del sistema. Tuttavia, al fine di evitare un comportamento imprevisto del sistema al verificarsi di un'eccezione inattesa, ai vettori di eccezione inutilizzati è di solito assegnato un indirizzo comune. A tale indirizzo viene posta una routine di eccezione "generalizzata". Questa routine potrebbe limitarsi a restituire il controllo al programma supervisore, per impedire che un programma di utente prosegua l'esecuzione e causi risultati imprevedibili. Per esempio, se un coprocessore in virgola mobile (*Floating Point CoProcessor:* FPCP) ed un'unità di gestione della memoria impaginata (*Paged Memory Management Unit:* PMMU) non fossero presenti nel sistema, allora i vettori da 47 a 58 potrebbero essere riempiti con l'indirizzo iniziale di una siffatta routine di gestione dell'eccezione.

All'inizializzazione, la tabella di vettori occupa le prime 1024 locazioni nella memoria. Infatti, il registro di base del vettore (*Vector Base Register:* VBR) contiene l'indirizzo iniziale della tabella di vettori ed il suo valore iniziale è $00000000. L'indirizzo di un vettore particolare è quindi individuato dalla CPU come:

indirizzo del vettore = (VBR) + <OFFSET>

dove OFFSET è il valore di offset specificato nella Tab. 10.8. Se una routine del sistema operativo cambia il contenuto del VBR usando l'istruzione MOVEC descritta nel par. 10.2, la tabella di vettori può essere trasferita nella memoria.

[7] Per inizializzare i valori, gli indirizzi potrebbero essere convertiti da circuiti speciali ad un indirizzo nella memoria a sola lettura, come avviene nel modulo MVME133. In ogni caso, la prime otto locazioni di byte non devono essere di tipo volatile quando la CPU accede ad esse per l'inizializzazione. La sequenza di reset viene avviata nel modulo MVME133 pigiando un pulsante che causa l'attivazione della linea di segnale $\overline{RESET}$ della CPU.

Tab. 10.8 *Tabella dei vettori di eccezione per l'MC68020. (Per gentile concessione di Motorola, Inc.)*

NUMERO/ I DEL VETTORE	OFFSET DEL VETTORE HEX	SPAZIO	ASSEGNAZIONE
0	000	SP	Reset: punt. stack interruz. iniziale
1	004	SP	Reset: contatore di programma iniziale
2	008	SD	Errore di bus
3	00C	SD	Errore d'indirizzo
4	010	SD	Istruzione illegale
5	014	SD	Divisione per zero
6	018	SD	Istruzione CHK, CHK2
7	01C	SD	Istruzioni cpTRAPcc, TRAPcc, TRAPV
8	020	SD	Violazione di privilegio
9	024	SD	Traccia
10	028	SD	Emulatore di linea 1010
11	02C	SD	Emulatore di linea 1111
12	030	SD	(Non assegnato, riservato)
13	034	SD	Violazione di protocollo del coprocessore
14	038	SD	Errore di formato
15	03C	SD	Interruzioni non inizializzata
16	040	SD	
intermedi			(Non assegnati, riservati)
23	05C	SD	
24	060	SD	Interruzione spuria
25	064	SD	Autovettore d'interruz. di livello 1
26	068	SD	Autovettore d'interruz. di livello 2
27	06C	SD	Autovettore d'interruz. di livello 3
28	070	SD	Autovettore d'interruz. di livello 4
29	074	SD	Autovettore d'interruz. di livello 5
30	078	SD	Autovettore d'interruz. di livello 6
31	07C	SD	Autovettore d'interruz. di livello 7
32	080	SD	
intermedi			Vettori d'istruzione TRAP 0-15
47	08C	SD	
48	0C0	SD	FPCP Salta o imposta su condiz. non ordinata
49	0C4	SD	FPCP Risultato inesatto
50	DC8	SD	FPCP Divisione per zero
51	0CC	SD	FPCP Underflow
52	0D0	SD	FPCP Errore di operando
53	0D4	SD	FPCP Overflow
54	0D8	SD	FPCP NAN segnalante
55	0DC	SD	(Non assegnato, riservato)
56	0E0	SD	PMMU Configurazione
57	0E4	SD	PMMU Operazione illegale
58	0E8	SD	PMMU Violazione del livello di accesso

Tab. 10.8 *Tabella dei vettori di eccezione per l'MC68020. (continuazione)*

NUMERO/ I DEL VETTORE	OFFSET DEL VETTORE HEX	SPAZIO	ASSEGNAZIONE
59	0EC	SD	(Non assegnati, riservati)
intermedi			
63	0FC	SD	
64	100	SD	Vettori definiti dall'utente (192)
intermedi			
266	3FC	SD	

SP = *Supervisor Program space:* spazio di programma di supervisore
SD = *Supervisor Data space:* spazio di dati di supervisore

Creazione della tabella dei vettori di eccezione e routine di gestione delle eccezioni. La CPU trasferisce automaticamente il controllo alla routine di gestione dell'eccezione definita dal vettore appropriato allorché si verifica un'eccezione. Quindi, la risposta a qualunque eccezione che non sia quella di reset non è determinata dalla CPU ma dipende interamente dall'operazione della routine. Queste routine di gestione delle eccezioni possono essere generalmente suddivise in due categorie: quelle fornite dal programma monitor e quelle create dal programmatore del sistema. Come esempi del primo tipo, le routine sono fornite dalla Motorola o da altre società come parte del software di sistema per elaborare le richieste di I/O ed altre operazioni riguardanti i dispositivi periferici standard quando si tratta di gestire le interruzioni. Tali routine saranno denotate qui come *routine standard.* Le eccezioni che indicano che un programma in modo utente ha generato un errore in genere causano la terminazione del programma stesso da parte del software di sistema, eventualmente con qualche indicazione sul terminale video in merito al tipo di errore. In sistemi con hardware o esigenze di tipo particolare, il programmatore di sistema deve creare e collaudare le routine di gestione delle eccezioni che sono pecuiari della particolare applicazione. Queste routine saranno denotate come gestori di eccezioni *speciali* nelle discussioni di questo capitolo.

Il monitor 133BUG descritto nel par. 5.1 fornisce un esempio di un programma supervisore con semplici capacità di gestione delle eccezioni. Oltre a gestire certe eccezioni in una maniera standard definita dalla Motorola, questo monitor consente di definire le routine speciali di gestione delle eccezioni ed i relativi vettori. Le eccezioni che sono gestite dalle routine standard del monitor sono quelle per l'istruzione illegale, la traccia, TRAP #15, e le interruzioni di livello 7. Le caratteristiche di traccia e di TRAP #15 del monitor sono state definite nel cap. 5 e sono state usate nei precedenti esempi di questo libro. L'interruzione di livello 7 è attivata allorché viene pigiato il pulsante ABORT del modulo MVME133 per terminare

l'esecuzione del programma e causare un ritorno al monitor. Tutte le altre eccezioni sono gestite da una routine generalizzata che termina il programma di utente e fa ritorno al monitor.

La routine d'inizializzazione. La routine d'inizializzazione prepara il sistema per l'esecuzione del programma supervisore. Questa preparazione include il caricamento degli indirizzi dei vettori nelle locazioni mostrate nella Tab. 10.8 e l'inizializzazione di qualsiasi dispositivo esterno. Quando la routine viene completata, il controllo può essere passato ad una routine di supervisore, che determina il programma applicativo da eseguire. Se non c'è un supervisore, il controllo viene passato direttamente al programma applicativo subito dopo il completamento dell'inizializzazione del sistema.

La routine d'inizializzazione per computer basati sull'MC68020 di solito svolge la serie di operazioni mostrate nella Tab. 10.9, sebbene molte varianti siano possibili per un'applicazione specifica. Dopo che il sistema è stato correttamente inizializzato, il puntatore di stack di utente, il registro di stato ed il contatore di programma sono predisposti in modo da passare il controllo al programma applicativo, che si presume che operi nel modo di utente nella sequenza mostrata nella Tab. 10.9. Come appare dalla tabella, il programma di supervisore controlla tutte le

Tab. 10.9 *Una tipica procedura d'inizializzazione del sistema.*

Operazione	Commenti
Inizializzazione dell'indirizzo per i vettori come richiesto.	Locazioni $0008-$03FC (indirizzi di 32 bit).
Caricamento di qualsiasi routine di supervisore necessaria.	Caricamento dall'unità a disco.
Inizializzazione del contenuto di ogni locazione di memoria richiesta dal supervisore.	Come richiesto.
Inizializzazione di tutti i servizi periferici per il sistema.	Come richiesto: istruzione RESET per dispositivi periferici ed altri comandi di controllo di I/O.
Inizializzazione dell'USP.	(USP) ← indirizzo di stack di utente.
Inserimento della word di formato sullo stack.	((SSP)) ← word di formato.
Inserimento dell'indirizzo iniziale sullo stack per un programma nel modo di utente.	((SSP)) ← inizio del programma.
Inserimento dello stato sullo stack per un programma nel modo di utente.	((SSP)) ← (SR) utente
Abilitazione del cache.	(CACR) = $00000001
Trasferimento del controllo ad un programma in modo utente.	RTE

operazioni del sistema e l'impostazione iniziale del puntatore di stack per il programma applicativo. Se l'utilizzazione della memoria è controllata anch'essa dal supervisore, allora i programmi applicativi sarebbero caricati nella memoria prima che il controllo sia passato all'utente.

10.3.2 Esempio d'inizializzazione

Il breve programma d'inizializzazione nella Fig. 10.8 è caricato con l'indirizzo iniziale appena sopra l'area di RAM del 133BUG, da $0000 a $3FFF. Quest'area nella memoria bassa contiene la tabella di vettori di eccezione ($0000-$03FF) ed altre informazioni necessarie per il funzionamento del programma di monitor. In termini della sequenza d'inizializzazione definita nella Tab. 10.9, questa routine specifica diversi vettori d'interruzione (livelli 0, 4, 5, 6 e 7), definisce l'indirizzo di TRAP #0, e passa il controllo ad un programma di utente alla locazione $10000. Gli altri passi d'inizializzazione devono essere completati dal monitor 133BUG prima che il programma della Fig. 10.8 possa essere eseguito. Qualsiasi eccezione non definita da questo programma d'inizializzazione viene gestita dalle routine standard di gestione delle eccezioni del monitor.

ESERCIZI

10.3.1 La sequenza di istruzioni

```
MOVE.W   #$0000,SR
JMP      UTENTE
```

potrebbe essere utilizzata per trasferire il controllo ad un programma alla locazione UTENTE. Perché è migliore il metodo che impiega l'istruzione RTE? Si consideri il caso in cui la memoria è segmentata (e protetta) in uno spazio di programma di supervisore ed in uno spazio di programma di utente.

10.3.2 Si progetti una routine di gestione dell'eccezione per rispondere ad un'interruzione di livello 7. Tale routine dovrebbe semplicemente visualizzare "Interruzione di livello 7" sullo schermo video dell'operatore allorché l'interruzione viene riconosciuta. L'indirizzo del vettore è $07C in un sistema di computer standard basato sull'MC68020. Se possibile, si provi la routine sul sistema che si sta utilizzando. Per causare l'interruzione di livello 7 nel modulo MVME133, l'operatore potrebbe, ad esempio, pigiare il pulsante di "aborto".

10.3.3 Si definiscano alcuni aspetti dell'utilità del registro di base del vettore. Si consideri dapprima il caso in cui il VBR è usato per rilocare la tabella di vettori di eccezione dopo l'inizializzazione del sistema. Successivamente si considerino i sistemi in cui è necessaria più di una tabella di vettori di eccezione.

10.3.4 Per il sistema di computer che il lettore sta utilizzando, si faccia stampare la tabella dei vettori di eccezione e si individuino tutte le eccezioni che sono gestite dal programma monitor o dal sistema operativo. Si faccia riferimento alla Tab. 10.8 per i nomi delle eccezioni specifiche.

```
                         1.          TTL     FIGURA 10.8
                         2.          LLEN    120
                         3.   *
                         4.   *   ESEGUE L'INIZIALIZZAZIONE DEL SISTEMA
                         5.   *
  # 00007000             6.   XPTERR  EQU    $7000          ;INDIRIZZO PER INTERRUZIONE SPURIA
  # 00007100             7.   IOXPT   EQU    $7100          ;INDIRIZZO PER INTERRUZIONE DI I/O
  # 00007200             8.   TMXPT   EQU    $7200          ;INDIRIZZO PER INTERRUZIONE DI TIMER
  # 00007300             9.   PEXPT   EQU    $7300          INDIRIZZO PER INTERRUZIONE DI ERRORE DI PARITA'
  # 00007400             10.  PFXPT   EQU    $7400          ;INDIRIZZO PER INTERRUZIONE NON MASCHERABILE
                         11.  *
  # 00007500             12.  EXEC    EQU    $7500          ;INDIRIZZO DI EXECUTIVE
                         13.  *                             ;CHIAMATA DI SUPERVISORE
                         14.  *
  # 00007000             15.  USERSTK EQU    $7000          ;STACK DI UTENTE
  # 00010000             16.  USERIN  EQU    $10000         ;INDIRIZZO INIZIALE DI UTENTE
                         17.  *
                         18.  *   CARICA VETTORI D'INTERRUZIONE DEL 68020
                         19.  *
00000060                 20.          ORG    $060
00000060  00007000       21.          DC.L   XPTERR         ;INTERRUZIONE SPURIA, LIVELLO 0
                         22.  *
00000070                 23.          ORG    $070
00000070  00007100       24.          DC.L   IOXPT          ;INTERRUZIONE DI I/O, LIVELLO 4
00000074  00007200       25.          DC.L   TMXPT          ;INTERRUZIONE DI TIMER, LIVELLO 5
00000078  00007300       26.          DC.L   PEXPT          ;INTERRUZIONE DI ERRORE DI PARITA', LIVELLO 6
0000007C  00007400       27.          DC.L   PFXPT          ;INTERRUZIONE NON MASCHERABILE, LIVELLO 7
                         28.  *
                         29.  *   CARICA VETTORI DI TRAPPOLA DEL 68020
                         30.  *
00000080                 31.          ORG    $080           ;TRAPPOLA #0
00000080  00007500       32.          DC.L   EXEC           ;VETTORE DI TRAPPOLA DI CHIAMATA DI SUPERVISORE
                         33.  *
                         34.  *   INIZIALIZZA UTENTE
                         35.  *
00004000                 36.          ORG    $4000
00004000  207C 00007000  37.          MOVEA.L #USERTSK,A0   ;LEGGE INDIRIZZO DI STACK UTENTE
00004006  4E7B 8800      38.          MOVEC  A0,USP         ;  E LO CARICA IN SP DI UTENTE
                         39.  *
                         40.  *   SALVA INDIRIZZO INIZIALE DI UTENTE E REGISTRO DI STATO
                         41.  *       SULLO STACK DI SUPERVISORE
                         42.  *
0000400A  3F3C 0000      43.          MOVE.W #0,-(SP)       ;WORD DI FORMATO
0000400E  2F3C 00010000  44.          MOVE.L #USERIN,-(SP)  ;PC DI UTENTE
00004014  3F3C 0000      45.          MOVE.W #0,-(SP)       ;MODO = UTENTE,
                         46.  *                             ;  MASCHERA D'INTERRUZIONE = 000
                         47.  *
                         48.  *   TRASFERISCE IL CONTROLLO AL PROGRAMMA DI UTENTE
                         49.  *
00004018  4E73           50.          RTE
0000401A                 51.          END
```

Fig. 10.8 *Esempio di routine d'inizializzazione.*

CAPITOLO 11

TECNICHE DI GESTIONE DELLE ECCEZIONI

In questo capitolo è discussa la capacità di gestione delle eccezioni della CPU MC68020. Sono disponibili eccezioni per garantire un trasferimento ordinato del controllo da un programma in corso di esecuzione al programma supervisore. Le eccezioni dell'MC68020 possono essere suddivise a grandi linee in quelle causate da un'istruzione, inclusa qualche condizione insolita che si presenta durante la sua esecuzione, e in quelle causate da eventi esterni. Le eccezioni della prima categoria sono denominate *trappole* e rappresentano condizioni eccezionali causate dal programma stesso e che sono rivelate dalla CPU. Gli eventi del sistema ed altre condizioni generate dall'hardware esterno rappresentano la seconda categoria di eccezioni. L'*interruzione* e l'*eccezione di errore di bus* sono due di tali eventi esterni riconosciuti dal processore. Quando viene rivelato un qualunque tipo di eccezione, il controllo della CPU viene passato alla routine di gestione dell'eccezione, che fa parte del sistema operativo o del programma di monitor. Queste routine normalmente sono eseguite nel modo di supervisore. Esse rappresentano la parte più fondamentale del software di sistema, che viene eseguito al termine dell'inizializzazione del computer. Si tratta delle routine che controllano direttamente la CPU quando si verifica un'eccezione. Questo capitolo descrive la risposta della CPU alle eccezioni, come pure le tecniche per creare routine di gestione delle eccezioni.

11.0 LE ECCEZIONI DELL'MC68020 E L'ELABORAZIONE DELLE ECCEZIONI

La Tab. 11.1 elenca le eccezioni dell'MC68020 e le loro cause. I tipi di eccezione possono essere suddivisi in quelle generate da istruzioni di trappola, verifica del programma, eccezioni speciali, condizioni di errore d'istruzione, errori e condizioni del sistema, incluse le eccezioni di coprocessore, ed interruzioni. Gli ultimi tre tipi di eccezioni risultano dall'azione o dalle operazioni della circuiteria esterna alla CPU. Tutte le altre eccezioni possono presentarsi durante la normale esecuzione di un programma.

Tab. 11.1 *Eccezioni dell'MC68020.*

Tipo	Descrizione
Istruzione di trappola	
TRAP #<N>; N = 0, 1, ..., 15	Sedici routine di trappola possono essere definite ed utilizzate da un programma.
Controlli di programma	
DIVS, DICSL, DIVU, DIVUL	Trappola quando il divisore è zero.
CHK, CHK2	Trappola se il valore del registro fuoriesce dai confini.
TRAPcc	Trappola su una di 16 condizioni aritmetiche.
TRAPV	Trappola su overflow; cioè, se V = {1}.
Eccezioni speciali	
Istruzione di trappola non implementata	Trappola se il codice operativo è {1010} o {1111} in assenza di coprocessore.
Eccezioni di traccia	Traccia dopo la singola istruzione o dopo il cambiamento del controllo di flusso.
Breakpoint (BKPT)	Trappola se si verifica un errore di bus durante l'esecuzione dell'istruzione BKPT.
Condizioni di errore d'istruzione	
Violazione di privilegio	Trappola se il programma in modo utente tenta di eseguire un'istruzione privilegiata.
Istruzione illegale	Trappola se la word di operazione di un'istruzione non è una valida configurazione di bit.
Errore di indirizzo	Trappola se il coprocessore tenta di prelevare un'istruzione ad un indirizzo dispari.
Errori e condizioni del sistema	
Errore di bus	Richiesta di eccezione generata esternamente.
Errore di formato	Trappola se l'errore si verifica durante RTE, CALLM o RTM, o cpRESTORE.*
Arresto (*Halt*)	Una condizione di doppio errore arresta il processore.
Eccezioni di coprocessore	Eccezioni causate dall'attività del coprocessore.
Sistema d'interruzione	
Autovettore	Vettorizzazione automatica per sette livelli di priorità.
Vettorizzato	192 vettori d'interruzioni definibili dall'utente; la priorità è determinata dal progetto hardware.

Nota: Le istruzioni CALLM e RTM non sono riconosciute dalla CPU MC68030.

La Fig. 11.1 illustra la sequenza di elaborazione per la maggior parte delle eccezioni, tranne quella di reset, che è stata descritta nel cap. 10. Come mostrato in Fig. 11.1(a), la CPU, nel momento in cui riconosce un'eccezione, entra nel modo di supervisore, sospende il tracciamento e determina il numero di vettore dell'eccezione. Questo numero ed il suo indirizzo di memoria sono mostrati nella Fig. 11.2 per ciascuna eccezione dell'MC68020. Dopo il riconoscimento dell'eccezione, la CPU salva le informazioni concernenti la particolare eccezione, creando un "frame di stack" sullo stack di supervisore. Tali informazioni comprendono almeno una word di formato ed i precedenti valori del contatore di programma e del registro di stato. Al termine di questa sequenza di elaborazione, il PC viene caricato con l'indirizzo del vettore della routine di gestione dell'eccezione ed il controllo della CPU viene passato a questa routine.

Salva (SR) internamente

Poni:
S = {1}
Bit di traccia = {0}

Modo di supervisore: traccia disattivata

Determina il numero del vettore

Per interruzioni vettorizzate, il numero del vettore è fornito dal dispositivo esterno: per tutte le altre eccezioni, esso è generato internamente.

Inserisci in stack di supervisore: word di formato, (PC), (SR) salvato

Crea il frame di stack. Salva l'indirizzo di ritorno e lo stato.

Carica l'indirizzo dell'eccezione: (PC) ← (indir. vettore)

Avvia l'esecuzione dell'istruzione nella routine di eccezione

Gestisce l'eccezione.

Routine di gestione dell'eccezione

RTE

Nota: l'elaborazione per il reset e certe altre eccezioni è lievemente diversa da quella mostrata.

(a)

RTE

Verifica la validità del frame di stack (word di formato)

VALIDO?

NO

ERRORE DI FORMATO

SI

Ripristina (SR) e (PC)

Continua l'elaborazione normale

Nota: Per certe eccezioni, vengono ripristinate informazioni ulteriori nei registri interni della CPU.

(b)

Fig. 11.1 *(a) Sequenza di elaborazione dell'eccezione al riconoscimento di un'eccezione. (b) ritorno dalla sequenza di eccezione.*

Tabella di vettori
(puntatori a routine di servizio)

OFFSET	31 — 0	NUMERO
VBR → $0	RESET (ISP)	0
$4	RESET (PC)	1
$8	BERR	2
$C	ERRORE D'INDIRIZZO	3
$10	ISTRUZIONE ILLEGALE	4
$14	DIVISIONE PER ZERO	5
$18	CHK, CHK2	6
$1C	TRAPV, TRAPcc, cpTRAPcc	7
$20	VIOLAZ. DI PRIVILEGIO	8
$24	TRACCIA	9
$28	EMULATORE DI LINEA A	10
$2C	EMULATORE DI LINEA F	11
$30	RISERVATO	12
$34	VIOLAZ. PROTOCOL. DI CP	13
$38	ERRORE DI FORMATO	14
$3C	I/O NON INIZIALIZZATO	15
$40	RISERVATO	16-23
$60	INTERRUZIONE SPURIA	24
$64	AUTOVETTORI 1-7	25-31
$80	TRAP #0-#15	32-47
$C0	SALTO DI FPCP	48
$C4	RISULT. INESATT. DI FPCP	49
$C8	DIVIS. PER ZERO DI FPCP	50
$CC	UNDERFLOW DI FPCP	51
$D0	ERR. OPERANDO DI FPCP	52
$D4	OVERFLOW DI FPCP	53
$D8	NAN SEGNALANTE FPCP	54
$DC	RISERVATO	55
$E0	CONFIGURAZ. DI PMMU	56
$E4	OPERAZ. ILLEG. DI PMMU	57
$E8	VIOLAZ. LIV. ACC. PMMU	58
$EC	RISERVATO	59-63
$100	VETTORI D'INTERRUZIONE DEL DISPOSITIVO DI I/O	64
:		:
$3FC		255

Routine di servizio
del sistema operativo

- Routine d'inizializzazione
- Routine del gestore di difetto
- Routine di errore d'indirizzo
- Debug monitor breakpoint
- Routine di divisione per zero
- Routine verifica del servizio
- Servizio TRAP condizionale
- Servizio violaz. di privilegio
- Debug monitor per traccia
- Routine di emulazione
- Emulazione di coprocessore
- Violaz. protocollo coprocess.
- Routine di errore di formato
- Interruzioni non inizializzate
- Servizio d'interruzione spuria
- Routine autovettorizzate
- Routine di servizio trappola
- Servizio di salto di FPCP
- Servizio risult. inesatto FPCP
- Serv. divis. per zero FPCP
- Routine di overflow di FPCP
- Serv. errore operando FPCP
- Servizio di overflow di FPCP
- Serv. NAN segnalante FPCP
- Serv. di configuraz. di PMMU
- Operazioni illegali di PMMU
- Violaz. liv. accesso di PMMU
- Routine di servizio d'interruzione 1
- Routine di servizio d'interruzione 192 … RTE

Note:
(1) L'offset del vettore è esadecimale; il numero del vettore è un valore decimale.
(2) CP = coprocessore; FPCP = coprocessore in virgola mobile;
PMMU = unità di gestione della memoria impaginata.

Fig. 11.2 *Assegnazione dei vettori. (Per gentile concessione di Motorola, Inc.)*

Quando la routine di gestione dell'eccezione viene completata eseguendo un'istruzione RTE (*ReTurn form Exception:* torna da eccezione), il controllo è restituito al programma che era in corso di esecuzione allorché l'eccezione si è verificata. La sequenza di ritorno è illustrata nella Fig. 11.1(b). Se il frame di stack ha un formato valido, i precedenti valori di (PC) e (SR) vengono ripristinati ed il puntatore di stack viene posto al valore che conteneva prima che si verificasse l'eccezione. Qualunque altra informazione nello stack sarà poi caricata nei registri interni della CPU. Tranne che per scopi di debugging, questa informazione non è normalmente d'interesse per un programmatore.

Il formato generale del frame di stack di eccezione per l'MC68020 è mostrato nella Fig. 11.3(a). Come mostrato nella Fig. 11.3(b), la lunghezza minima del frame di stack è di 4 word di 16 bit se nessuna ulteriore informazione sullo stato del processore viene posta sullo stack allorché l'eccezione è riconosciuta. Certe eccezioni richiedono informazioni addizionali, che saranno descritte in questo capitolo quando si discuteranno le eccezioni specifiche. La Fig. 11.3(c) mostra un frame di stack di 6 word come esempio. Le word di formato associate a ciascun frame di stack sono presentate nella Fig. 11.3(d).

Questo capitolo descrive la maggior parte delle eccezioni dell'MC68020. Nel par. 11.1 è discussa la gestione delle eccezioni da parte dei sistemi operativi e dei programmi di monitor. Dal par. 11.2 al par. 11.5 saranno definite le eccezioni specifiche e saranno presentati alcuni esempi del loro impiego. La gestione delle interruzioni da parte dell'MC68020 sarà descritta nel par. 11.6. I formati dei frame di stack di eccezione saranno riepilogati nel par. 11.7. In tale paragrafo sarà anche trattato il metodo di priorità adottato dall'MC68020 quando più eccezioni si presentano simultaneamente.

11.1 GESTIONE DELLE ECCEZIONI DA PARTE DEI SISTEMI OPERATIVI E DEI PROGRAMMI DI MONITOR

Qualsiasi programma supervisore fornito dalla Motorola o da altre società per sistemi basati sull'MC68020 avrà un certo numero di routine di gestione delle eccezioni come parte del software di sistema. In questo paragrafo, tali routine saranno definite gestori "standard" di eccezione. Il loro scopo è quello di gestire le eccezioni che sono definite per il sistema di computer indipendentemente dall'applicazione specifica. Un'altra classe di routine di gestione delle eccezioni — denotate come "speciali" in questo paragrafo — dev'essere fornita dal programmatore di sistema, che provvede ad adattare il sistema di computer alle particolari esigenze dell'applicazione specifica. Come esempio, il sistema operativo VERSAdos della Motorola dispone di un certo numero di routine standard di gestione delle eccezioni, ma consente anche di aggiungere gestori speciali al sistema operativo di base. Il monitor BUG133 include anch'esso diversi gestori standard come parte del suo programma contenuto in ROM.

15 12 0
SP → Registro di stato
Contatore di programma
Formato | Offset del vettore
Ulteriori informazioni sullo stato del processore
(2, 6, 12 o 42 word, se necessario)

15 0
SP → Registro di stato
+ $02 Contatore di programma
+ $06 0 0 0 0 | Offset del vettore

15 0
SP → Registro di stato
+ $02 Contatore di programma
+ $06 0 0 1 0 | Offset del vettore
+ $08 Indirizzo dell'istruzione

Formato	Tipo di frame
0000	Formato corto (4 word)
0001	Scarto (4 word)
0010	Eccezione di istruzione (6 word)
0011-0111	(Indefinito, Riservato)
1000	Difetto di bus dell'MC68020 (29 word)
1001	Istruzione intermedia di coprocessore (10 word)
1010	Difetto di bus corto dell'MC68020 (16 word)
1011	Difetto di bus lungo dell'MC68020 (46 word)
1100-1111	(Indefinito, Riservato)

Fig. 11.3 *(a) Frame di stack di eccezione dell'MC68020; (b) frame di stack di quattro word; (c) frame di stack di sei word; (d) codici di formato per i frame di stack di eccezione. (Per gentile concessione di Motorola, Inc.)*

11.1.1 Gestori di eccezione standard

I sistemi operativi per computer basati sull'M68020 dispongono di una capacità di gestione delle eccezioni standard che può essere distinta in due categorie principali. La prima categoria include quelle routine che forniscono un servizio ad un programma. I servizi comprendono le operazioni d'ingresso/uscita, la rivelazione degli errori, ed altri che sono normalmente gestiti dal sistema operativo. Un servizio specifico è richiesto dall'istruzione

```
TRAP        #<N>
```

nella maggior parte dei sistemi basati sull'MC68020. La costante N serve a definire una delle 16 possibili richieste avanzate al sistema operativo; N = 0, 1, 2, ..., 15. Diversi esempi nel cap. 7 mostravano l'impiego dell'istruzione TRAP per richiedere i servizi del supervisore. La seconda categoria di routine standard delle eccezioni fornite dal sistema operativo include routine per il controllo dell'attività dei dispositivi periferici collegati al computer. Tali dispositivi comprendono le unità di memorizzazione su disco e i terminali di operatore.

11.1.2 Gestori di eccezione speciali

In molte applicazioni, durante l'attività del sistema di computer si presentano delle condizioni particolari che richiedono apposite routine speciali di gestione delle eccezioni per soddisfare i requisiti dell'applicazione. Queste condizioni non possono essere previste dal produttore del sistema di computer e del suo sistema operativo. Pertanto, le suddette routine speciali devono essere progettate e collaudate in base alle specifiche esigenze dell'applicazione. Queste routine saranno aggiunte al sistema operativo allorché saranno state completate. Il programma supervisore per il computer diverrà così un sistema operativo modificato, che includerà sia le routine standard che quelle speciali per la gestione delle eccezioni.[1]

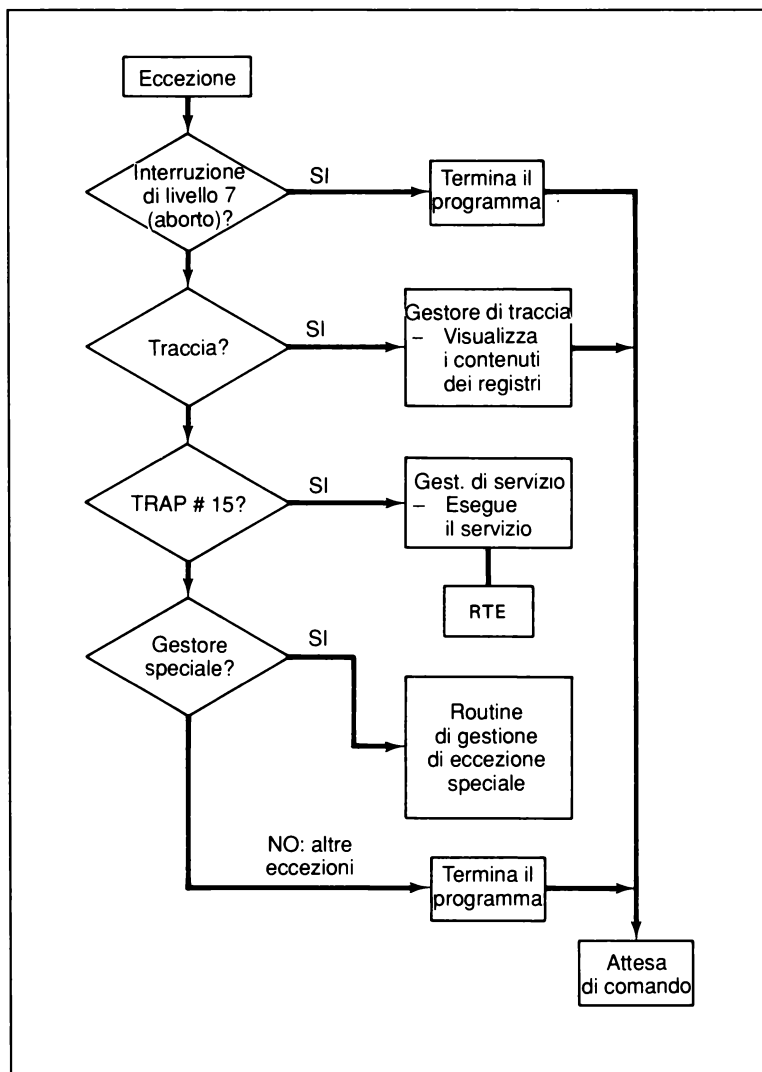
11.1.3 Gestione delle eccezioni da parte del monitor 133BUG

La risposta ad un'eccezione da parte del monitor 133BUG è mostrata nella Fig. 11.4. Come descritto in precedenza nel cap. 10.3, il programma del monitor gestisce l'interruzione di livello 7, la traccia e le eccezione di TRAP #15. Altre eccezioni non gestite da routine speciali fanno sì che il monitor termini prematuramente il programma che ha causato l'eccezione. SI possono creare facilmente delle routine speciali per la gestione delle eccezioni in sistemi di computer che dispongano del monitor 133BUG. La tecnica di definire un indirizzo nella tabella dei vettori di eccezione, per puntare ad una routine speciale, è stata già discussa nel cap. 10.

---

[1] Il metodo per aggiungere una routine speciale ad un sistema operativo dipende completamente dal progetto del sistema operativo. I dettagli specifici saranno forniti nel manuale per l'utente del sistema operativo considerato.

Fig. 11.4
Gestione delle eccezioni da parte del monitor 133BUG.

Eccezione
Interruzione di livello 7 (aborto)?
SI
Termina il programma
Traccia?
SI
Gestore di traccia
– Visualizza i contenuti dei registri
TRAP # 15?
SI
Gest. di servizio
– Esegue il servizio
RTE
Gestore speciale?
SI
Routine di gestione di eccezione speciale
NO: altre eccezioni
Termina il programma
Attesa di comando

11.1.1

ESERCIZIO

Si scelga un sistema operativo d'interesse e si descriva il modo in cui il software di sistema provvede all'ingresso/uscita per la gestione degli errori. Se tale sistema operativo non è stato progettato per operare su un computer basato sull'MC68020, si descriva il modo in cui le eccezioni gestite da esso potrebbero essere implementate in un sistema della Motorola.

11.2 ECCEZIONI CAUSATE DA ISTRUZIONI DI TRAPPOLA E DA VERIFICHE DEL PROGRAMMA

Varie eccezioni, denominate *trappole*, sono disponibili al programmatore per controllare il funzionamento di un programma. Tali eccezioni sono elencate nella Tab. 11.2 e comprendono eccezioni che possono essere causate da:

(a) Istruzioni di trappola; oppure:
(b) Un'istruzione per la verifica delle condizioni del programma.

Un'istruzione di trappola serve solitamente a restituire il controllo al programma supervisore quando è richiesto qualche tipo di servizio di supervisore. Le istruzioni per verificare il funzionamento di un programma sono usate per effettuare vari test su condizioni aritmetiche o sull'intervallo di validità di un valore contenuto in un registro. Se viene rivelata una condizione di errore, allora scatta una trappola. Quando si verifica una di queste trappole, la routine di gestione delle eccezioni può tentare di porre rimedio alla condizione che ha causato la trappola e riportare il controllo al programma che l'ha generata. Tuttavia, nella maggior parte dei casi che riguardano programmi applicativi, il programma responsabile viene terminato allorché un'istruzione di verifica del programma fa scattare una trappola. La risposta precisa dipende completamente dal progetto del sistema operativo e dalle sue routine di gestione delle eccezioni.

Tab. 11.2 *Istruzioni di trappola e verifiche di programma.*

Tipo di trappola	Causa della trappola	Indirizzo del vettore	Commenti
Istruzione TRAP	Esecuzione normale	\$080-\$0BC	Metodo tipico per chiamare il programma di supervisore
Verifiche di programma			
DIVS, DIVSL, DIVU DIVUL	Il divisore è zero	\$014	Verifica aritmetica
CHK, CHK2	Il valore in un registro è fuori dei confini	\$018	Verifica dei confini
TRAPcc	È stata rivelata la condizione logica "cc"	\$01C	Possibilità di selezione di una tra 16 condizioni
TRAPV	Esecuzione con V = {1}	\$01C su overflow aritmetico	Trappola

11.2.1 L'istruzione TRAP

L'esecuzione dell'istruzione TRAP col formato

TRAP #<vettore>

per prima cosa produce l'inserimento nello stack di supervisore di una word di formato e dei contenuti del contatore di programma e del registro di stato, in quest'ordine. Dopodiché, l'elaborazione inizia dall'indirizzo specificato nella locazione del vettore. Il valore <vettore> è un intero nell'intervallo 0-15 ed è usato per calcolare l'indirizzo del vettore esadecimale, come segue:

Indirizzo del vettore = 80_{16} + 4 * <vettore>

L'elaborazione dell'eccezione inizia dalla locazione caricata nel contatore di programma con:

(PC) ← (indirizzo del vettore)

dove ciascun indirizzo è lungo 32 bit. Per comodità, gli indirizzi dei vettori sono elencati nella Tab. 11.3. Queste locazioni devono essere inizializzate prima che TRAP venga eseguita.

Tab. 11.3
Indirizzi dei vettori di trappola.

Istruzione TRAP #<N>	Indirizzo del vettore (esadecimale)
TRAP #0	80
TRAP #1	84
TRAP #2	88
TRAP #3	8C
TRAP #4	90
TRAP #5	94
TRAP #6	98
TRAP #7	9C
TRAP #8	A0
TRAP #9	A4
TRAP #10	A8
TRAP #11	AC
TRAP #12	B0
TRAP #13	B4
TRAP #14	B8
TRAP #15	BC

Nota: Il numero di vettore è decimale.

L'istruzione TRAP ha numerose applicazioni. Per un programma operante nel modo di utente, la sua esecuzione può riportare il controllo al programma supervisore, nella locazione della routine di trappola designata. Le 16 trappole possibili consentono ad un programma in modo utente di chiamare il supervisore per l'elaborazione. La routine di supervisore dev'essere eseguita nel modo di supervisore del processore. Per esempio, una chiamata mediante l'istruzione TRAP potrebbe essere usata per l'ingresso o l'uscita di dati tramite dispositivi periferici controlati dal supervisore. In effetti, la trappola è un'interruzione software. Questo meccanismo potrebbe essere utilizzato nelle operazioni di debugging per simulare le interruzioni. L'istruzione TRAP è anche un metodo per restituire il controllo al modo di supervisore al termine del compito dell'applicazione. Altri esempi dell'impiego di TRAP sono stati forniti nel cap. 7.

Esempio 11-1

La Fig. 11.5 contiene un segmento di programma in un executive (supervisore) in tempo reale, che può essere chiamato dall'utente tramite un'istruzione

```
TRAP        #0
```

L'informazione è codificata dopo la TRAP usando un'istruzione DC.W per definire un intero di 16 bit (numero di funzione) che seleziona il particolare servizio esecutivo richiesto dall'utente. Il numero di funzione è un intero di valore 0, 1, 2 o 3.

Il segmento che inizia da EXEC disabilita dapprima tutte le interruzioni chiamando la subroutine DIALL (non mostrata), dopodiché salva i registri di utente sullo stack. Successivamente viene reperito dallo stack l'indirizzo di ritorno entro il programma di utente. Questo indirizzo punta alla locazione che segue l'istruzione TRAP che contiene il numero di funzione di 16 bit. Questo numero viene convalidato ed una tabella di salto viene usata per ottenere l'indirizzo della routine di executive selezionata. Tale indirizzo viene posto sullo stack e viene eseguito un ritorno per trasferire il controllo alla funzione selezionata. I registro d'indirizzo A0 contiene l'indirizzo del valore di dati che segue l'istruzione TRAP #0 che la richiesto la funzione di executive.

L'elaborazione dell'errore inizia quando viene rivelato un numero di funzione "illegale" (cioè, non ammesso). L'indirizzo dell'errore nel programma di utente ed il codice dell'errore vengono salvati prima che il controllo sia trasferito alla routine di elaborazione dell'errore.

```
                              1.          TTL     FIGURA 11.5
                              2.          LLEN    120
00000080                      3.          ORG     $080
00000080  00007500            4.          DC.L    EXEC          ;INIZIALIZZA TRAPPOLA 0
                              5.          ORG     $7500
                              6.   *
    * 00000010                7.   EEXEC  EQU     $10           ;CODICE DI ERRORE IN EXECUTIVE
    * 00004100                8.   DIALL  EQU     $4100         ;ROUTINE PER DISABILITARE LE INTERRUZIONI
    * 00004200                9.   DISP   EQU     $4200         ;ROUTINE PER LA DISTRIBUZIONE (DISPATCHING)
    * 00004300               10.   INTSCH EQU     $4300         ;ROUTINE PER SALVARE I TASK INTERROTTI
    * 00004400               11.   SCHED  EQU     $4400         ;ROUTINE PER ESEGUIRE LO SCHEDULING
    * 00004500               12.   RTERRX EQU     $4500         ;ROUTINE PER L'ELABORAZIONE DELL'ERRORE
                             13.   *
                             14.   *          EXEC
                             15.   *
                             16.   *          QUESTA ROUTINE DETERMINA LA FUNZIONE DI EXECUTIVE RICHIESTA
                             17.   *          E TRASFERISCE IL CONTROLLO AD ESSA. EXEC INSERISCE 15 REGISTRI
                             18.   *          NELLO STACK ED IMPIEGA UNA TABELLA DI SALTO PER PASSARE IL
                             19.   *          CONTROLLO ALLA FUNZIONE APPROPRIATA. ESSA PASSA IN A0 L'INDIRIZZO
                             20.   *          DEL DATO DELLA RICHIESTA DI FUNZIONE E PASSA IN D1 IL CODICE DELLA
                             21.   *          FUNZIONE * 4.
                             22.   *
                             23.   *          EXEC VIENE ESEGUITA A LIVELLO DI SISTEMA ED E' CHIAMATA
                             24.   *          COME INDICATO DI SEGUITO:
                             25.   *                  TRAP          #0
                             26.   *                  DC.W          #FUNZIONE (0 = DISP, 1 = SCHED, 2 = TMSCH, 3 = INTSCH)
                             27.   *                  DC.           DATI ADDIZIONALI, DEFINITI DA CIASCUNA FUNZIONE
                             28.   *
                             29.   *          EXEC RICHIAMA LE SEGUENTI ROUTINE:
                             30.   *                  DIALL  - DISABILITA INTERRUZIONI
                             31.   *                  DISP   - DISPATCHER
                             32.   *                  INTSCH - SCHEDULATORE DI TASK INTERROTTO
                             33.   *                  RTERRX - USCITA DI ERRORE RTE
                             34.   *                  SCHED  - SCHEDULATORE DI TASK
                             35.   *
                             36.   *          EXEC USA I SEGUENTI DATI:
                             37.   *                  EEXEC  - CODICE DI ERRORE IN EXEC (CIOE', FUNZIONE ILLEGALE)
                             38.   *                  ERRAD  - INDIRIZZO DI ERRORE
                             39.   *                  ERRCD  - CODICE DI ERRORE
                             40.   *                  EXECTB - TABELLA DI SALTO DI CONTROLLO DELL'EXECUTIVE
                             41.   *                  EXTBLN - LUNGHEZZA DELLA TABELLA DI CONTROLLO DELL'EXECUTIVE
                             42.   *
00007500  6100 CBFE          43.   EXEC   BSR     DIALL         ;DISABILITA LE INTERRUZIONI
00007504  48E7 FFFE          44.          MOVEM.L D0-D7/A0-A6,-(A7)    ;SALVA I REGISTRI
00007508  206F 003E          45.          MOVE.L  (62,A7),A0    ;LEGGE IL PC (PUNTA AL DATO DELLA RICHIESTA
                             46.   *                            ;  DI FUNZIONE CHE SEGUE LA CHIAMATA DI TRAP #0)
0000750C  4281               47.          CLR.L   D1
0000750E  3218               48.          MOVE.W  (A0)+,D1      ;LEGGE IL NUMERO DI FUNZIONE RICHIESTA
00007510  0C41 0004          49.          CMPI.W  #EXTBLN,D1    ;LA FUNZIONE E' AMMESSA?
00007514  6C 0E              50.          BGE.S   EXEC10        ;NO, ELABORA L'ERRORE
00007516  43F9 0000753E      51.          LEA     EXECTB,A1     ;ALTRIMENTI, LEGGE L'INDIRIZZO DELLA FUNZIONE
0000751C  E589               52.          LSL.L   #2,D1         ;  RICHIESTA
0000751E  D3C1               53.          ADD.L   D1,A1         ;(NUM. FUNZIONE * 4 + INDIRIZZO DI TABELLA)
00007520  2F11               54.          MOVE.L  (A1),-(A7)    ; E LO PONE SULLO STACK
                             55.   *
                             56.   *          ESEGUE IL 'RETURN' ALLA FUNZIONE IL CUI INDIRIZZO E' STATO POSTO SULLO STACK

Quelo ...A68K K6.2J2 June 29, 1987 ...Run on Sep 11, 1988 14:42:38...Page 2
                             57.   *
00007522 4E75                58.          RTS
                             59.   *
                             60.   *   ELABORAZIONE DELLA RIVELAZIONE DELL'ERRORE
                             61.   *
00007524  4CDF 7FFF          62.   EXEC10 MOVEM.L (A7)+,D0-D7/A0-A6  ;RIPRISTINA I REGISTRI
00007528  23EF 0002          63.          MOVE.L  (2,A7),ERRAD      ;INSERISCE NELLO STACK L'INDIRIZZO DI ERRORE
          00007552
00007530  23FC 00000010      64.          MOVE.L  #EEXEC,ERRCD      ;MEMORIZZA IL CODICE DI ERRORE
          0000754E
0000753A  4EF8 4500          65.          JMP     RTERRX            ;USCITA DI ERRORE RTE
                             66.   *
                             67.   *   TABELLA DI SALTO DI CONTROLLO DELL'EXECUTIVE E ALTRI DATI
                             68.   *
     * 00000004              69.   EXTBLN EQU     4             ;LUNGHEZZA DELLA TABELLA DI CONTROLLO DELL'EXECUTIVE
0000753E  00004200           70.   EXECTB DC.L    DISP          ;DISTRIBUTORE (DISPATCHER)
00007542  00004400           71.          DC.L    SCHED         ;SCHEDULATORE
00007546  00004400           72.          DC.L    SCHED         ;SCHEDULATORE TEMPORIZZATO
0000754A  00004300           73.          DC.L    INTSCH        ;SCHEDULATORE DI TASK INTERROTTO
0000754E  <4>                74.   ERRCD  DS.L    1             ;CODICE DI ERRORE
00007552  <4>                75.   ERRAD  DS.L    1             ;INDIRIZZO DI ERRORE
00007556                     76.          END
```

Fig. 11.5 *Esempio di impiego di TRAP.*

11.2.2 Trappola di divisione per zero

Certi errori aritmetici in un programma applicativo possono essere rivelati ed "intrappolati" dalla CPU. In particolare, l'esecuzione di un'istruzione di divisione con un divisore nullo causa automaticamente una trappola attraverso l'indirizzo di vettore $14. La maggior parte dei sistemi operativi pone termine al programma che ha causato questa trappola, poiché un'ulteriore elaborazione aritmetica dopo una divisione per zero è raramente desiderata dal programmatore. Quindi la routine di gestione della trappola indica solitamente il tipo di errore sul terminale dell'operatore e ritrasferisce il controllo al sistema operativo anziché al programma "intrappolato". Il programma operativo potrà quindi pianificare l'esecuzione di un altro programma. L'esempio 11.2 illustra il modo in cui una routine di gestione della trappola potrebbe consentire ad un programma applicativo di riprendersi dalla trappola di divisione per zero.

Questo tipo di trappola produce la creazione di un frame di stack di sei word, che include l'indirizzo dell'istruzione che ha causato la trappola. Il valore del PC sullo stack punta all'istruzione successiva da eseguire se viene eseguita l'istruzione RTE nella routine di gestione della trappola per restituire il controllo al programma che ha causato la trappola. Questo tipo di frame di stack è stato illustrato nella Fig. 11.3.

Esempio 11-2

La Fig. 11.6 (a pagina seguente) illustra un esempio di elaborazione di una trappola di divisione per zero. La prima sezione memorizza l'indirizzo della routine di gestione della trappola nella locazione del vettore di trappola in $14. Successivamente, un breve segmento di programma in $10000 produce una divisione per zero allo scopo di valutare la routine di trappola. Infine, la routine del gestore della trappola inizia in $5300 ed assegna al quoziente un valore particolare basato sul valore del dividendo, che può essere l'intero più positivo o più negativo ammissibile.

11.2.3 Le istruzioni CHK e CHK2

L'MC68020 ha due istruzioni per consentire la verifica che un valore in un registro sia compreso nell'appropriato intervallo numerico. L'istruzione CHK esamina il valore in un registro di dati per verificare che sia compreso in un intervallo da 0 ad un confine superiore specificato. Una seconda istruzione, CHK2, verifica che il valore contenuto in un registro d'indirizzo sia compreso tra due confini. Per ciascuna istruzione, una trappola scatta se il valore nel registro esaminato non rispetta i limiti. CHK2 è analoga all'istruzione CMP2 presentata nel cap. 9, ma quest'ultima non causa una trappola se il valore del registro è al di fuori dell'intervallo specificato.

```
                             1.         TTL     FIGURA 11.6
                             2.         LLEN    100
    * 00000063               3.  .RETURN EQU    $0063
00000014                     4.         ORG     $14
                             5.  *
                             6.  *  PREDISPONE IL VETTORE DI TRAPPOLA DI DIVISIONE PER ZERO
                             7.  *
00000014 00005300            8.         DC.L    ZDIV            ;ROUTINE DI TRAPPOLA
                             9.  *                              ; DI DIVISIONE PER ZERO
                            10.  *
00010000                    11.         ORG     $10000
                            12.  *
                            13.  *  ESEGUE UNA DIVISIONE PER ZERO, CON:
                            14.  *      INPUT:  (D0.W) = DIVISORE (AZZERATO PER IL TEST)
                            15.  *              (D1.L) = DIVIDENDO
                            16.  *      OUTPUT: (D1.W) = MASSIMO VALORE, IN BASE AL SEGNO
                            17.  *                       DEL DIVIDENDO
                            18.  *
00010000 4280               19.         CLR.L   D0              ;PONE IL DIVISORE A ZERO
00010002 83C0               20.         DIVS    D0,D1           ;DIVIDE PER D0
                            21.  *
00010004 4E4F               22.         TRAP    #15             ;RITORNA AL MONITOR
00010006 0063               23.         DC.W    .RETURN
                            24.  *
                            25.  ************************************************************************
                            26.  *
                            27.  *      GESTORE DI TRAPPOLA DI DIVISIONE PER ZERO
                            28.  *
00005300                    29.         ORG     $5300
00005300 4A81               30.  ZDIV   TST.L   D1
00005302 6B00 000C          31.         BMI     ZDIV01          ;SE IL DIVIDENDO E' POSITIVO,
00005306 223C 00007FFF      32.         MOVE.L  #$7FFF,D1       ;  RITORNA CON D1 = MASSIMO
0000530C 6000 0008          33.         BRA     RTN             ;  VALORE POSITIVO
                            34.  *
                            35.  *
00005310 223C 00008000      36.  ZDIV01 MOVE.L  #$8000,D1       ;ALTRIMENTI RIPORTA IL MASSIMO
                            37.  *                              ;  VALORE POSITIVO
00005316 4E73               38.  RTN    RTE
00005318                    39.         END
```

Fig. 11.6 *Esempio di trappola di divisione per zero. (Esempio 11-2)*

L'istruzione CHK. L'istruzione di verifica che il registro rispetti i confini (*CHecK register against bounds:* CHK) ha la forma simbolica:

CHK.<l_1> <EA>,<Dn>

dove <EA> è designato da una modalità d'indirizzamento di dati. Ciò consente tutti i modi d'indirizzamento, tranne quello diretto di registro d'indirizzo. La lunghezza <l_1> può essere di word (<l_1> = W) o di longword (<l_1> = L). Questa istruzione determina se il valore contenuto in <Dn> è compreso tra 0 ed il valore contenuto in <EA> e causa una trappola se tale valore non appartiene a questo intervallo. Il confine superiore contenuto in (EA) è trattato come un intero di 16 bit in complemento a 2. L'operazione è la seguente:

IF $0 < (Dn)[l_1] \leq (EA)$, THEN continua
ELSE *trappola* e
poni N = {1} se $(Dn)[l_1] < 0$ oppure
poni N = {0} se $(Dn)[l_1] > (EA)$

La routine di eccezione inizia dalla locazione all'indirizzo $18 se la trappola scatta. Di solito l'istruzione CHK viene posta in un programma dopo il calcolo dell'offset o di un valore di indice per garantire che i limiti del valore non siano superati. Ciò facilita la verifica che un indirizzo di array rientri nelle dimensioni dell'array quando s'impiega il modo d'indirizzamento indiretto di registro d'indirizzo con indicizzazione per individuare gli elementi dell'array. Per esempio, la sequenza

```
CHK.W     #99,D1
MOVE.B    (0,A1,D1.W),D2
```

farebbe scattare una trappola se (D1)[15:0] superasse il valore 99 decimale. Se A1 contenesse l'indirizzo di base dell'array di 100 elementi lunghi ciascuno 1 byte, allora l'intervallo d'indirizzamento dell'istruzione MOVE sarebbe limitato ai valori entro l'array. Nel FORTRAN, l'indirizzamento degli array non è sempre protetto in questa maniera, mentre la verifica dei confini dell'array è di solito disponibile in altri linguaggi come il Pascal.[2]

Altri impieghi dell'istruzione CHK includono il test dello spazio usato da uno stack o la protezione dello spazio dei dati di un programma dall'accesso da parte di altri programmi. Per queste applicazioni, un valore in un registro d'indirizzo deve essere trasferito ad un registro di dati, al fine di eseguire la verifica dei confini.

L'istruzione CHK2. L'istruzione CHK2 (*CHecK register against bounds:* verifica che il registro rispetti i confini) ha il seguente formato in linguaggio assembler:

```
CHK2.<l>    <EA>,Rn
```

dove <l> = B, W o L, mentre <EA> definisce l'indirizzo nella memoria dei confini impiegati per la verifica del valore contenuto nel registro Rn, che può essere un qualunque registro d'indirizzo o di dati. Un valore designato in un registro d'indirizzo come lungo un byte o una word viene esteso di segno a 32 bit per il confronto. L'indirizzo effettivo <EA> dev'essere definito da un modo di controllo dell'indirizzamento, per cui sono proibiti i modi d'indirizzamento diretto di registro, con predecremento o con postincremento ed immediato.

Nella memoria, alla locazione designata da <EA>, il confine inferiore occupa un byte, una word o una longword in accordo con la specificazione della lunghezza <l>. Il confine superiore deve seguire nella locazione successiva di lunghezza apporpriata. Il confronto è:

$$(\langle EA \rangle) \leq (Rn) \leq (\langle EA + k \rangle)$$

dove k = 1, 2 o 4 per operandi di byte, word o longword, rispettivamente.

---

[2] Il compilatore determina la disponibilità della verifica dei confini per un programma in esecuzione. Tale verifica è una caratteristica opzionale di alcuni compilatori.

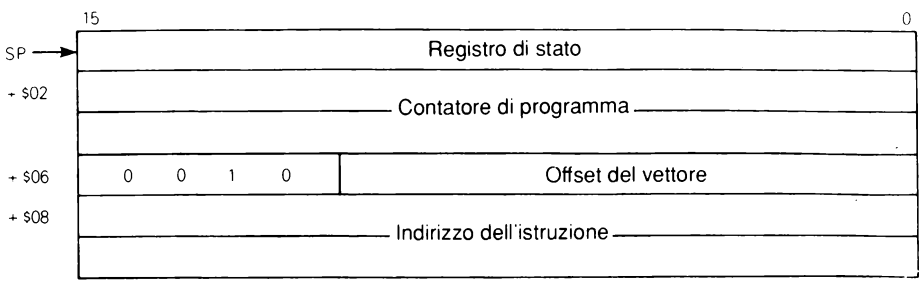
Quando l'istruzione viene eseguita, i codici di condizione sono impostati come per l'istruzione CMP2, ma scatta una trappola se il valore del registro fuoriesce dall'intervallo. A parte la trappola generata da una condizione di fuori-intervallo, CHK2 opera come l'istruzione CMP2 descritta nel par. 9.3. L'esempio 9.8 mostrava l'impiego dell'istruzione CMP2 per la verifica dei confini di un valore contenuto in un registro. L'istruzione CHK2 può essere impiegata nella medesima maniera. Tuttavia, una condizione di fuori-intervallo causa una trappola usando l'indirizzo del vettore alla locazione $018. La routine di gestione dell'eccezione individuata da quel vettore deve determinare l'azione appropriata quando il valore del registro da esaminare non rientra nell'intervallo.

Esempio 11-3

Le istruzioni CHK e CHK2 causano entrambe una trappola usando il vettore alla locazione $018 quando il valore del registro da verificare non rientra nei limiti dell'intervallo. Se la routine di gestione dell'eccezione deve distinguere tra un'eccezione di CHK ed una di CHK2, essa può determinare il tipo dell'eccezione che si è verificata soltanto esaminando la word di operazione per l'istruzione che ha causato l'eccezione. La locazione di questa istruzione viene posta nel frame di stack di eccezione nel momento in cui l'eccezione viene elaborata.

La Fig. 11.7(a) mostra il frame di stack di sei word creato quando un'istruzione CHK o CHK2 causa un'eccezione. Oltre a (SR), (PC) ed alla word di formato, lo stack contiene l'indirizzo dell'istruzione che ha causato l'eccezione. Tale indirizzo ha un offset di +08 byte dalla locazione indicata da (SP) stesso. Quindi l'istruzione viene facilmente esaminata usando l'indirizzamento indiretto della memoria. I formati del linguaggio-macchina per le istruzioni CHK e CHK2 sono mostrati nella Fig. 11.7(b).

Il programma nella Fig. 11.8 è un esempio di routine di gestione delle eccezioni. Esso definisce dapprima l'indirizzo del vettore per la routine di gestione delle eccezioni CHK e CHK2 all'indirizzo CHKTRAP ($5400 nell'esempio) allorché viene assemblato e caricato nella memoria. Se scatta una trappola, l'istruzione di test del campo di bit (*Bit Field TeST:* BFTST) esamina i bit del codice operativo dell'istruzione che ha causato l'eccezione ed assegna i valori ai codici di condizione. Dopodiché, le istruzioni di salto avviano l'esecuzione del segmento di codice appropriato. Per un'eccezione di CHK2, viene eseguito il segmento definito all'indirizzo CHK2SUB. Nell'esempio, l'elaborazione consiste semplicemente in una restituzione del controllo al programma del monitor. Se si presenta un'eccezione di CHK, allora viene eseguito il segmento di codice in CHKSUB.

	15				0
SP →	Registro di stato				
+ $02	Contatore di programma				
+ $06	0	0	1	0	Offset del vettore
+ $08	Indirizzo dell'istruzione				

(a) *Frame di stack.*

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	Registro Dn			Dimensione			Indirizzo effettivo Modo			Registro		

(b) *Istruzione CHK.*

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	Dimensione		0	1	1	Indirizzo effettivo Modo			Registro		
D/A	Registro			1	0	0	0	0	0	0	0	0	0	0	0

(b) *Istruzione CHK2.*

Fig. 11.7 *(a) Frame di stack per le istruzioni CHK e CHK2. (b) Formati delle istruzioni.*

```
                              1.          TTL     FIGURA 11.8
                              2.          LLEN    100
   # 00000063                 3.  .RETURN EQU     $0063          ;MONITOR
                              4.  *
                              5.  *   DEFINISCE L'INDIRIZZO DEL VETTORE
                              6.  *
00000018                      7.          ORG     $018
00000018  00005400            8.          DC.L    CHKTRAP        ;INDIRIZZO DEL VETTORE
                              9.  *
00005400                     10.          ORG     $5400
                             11.  *
                             12.  *   DETERMINA LA CAUSA DELLA TRAPPOLA,
                             13.  *   ESEGUE LA ROUTINE APPROPRIATA
                             14.  *   E TORNA AL MONITOR.
                             15.  *
                             16.  *       SE CHK2: ESEGUE CHK2SUB E RITORNA
                             17.  *       SE CHK:  ESEGUE CHKSUB E RITORNA
                             18.  *
00005400  E8F7 0004 0161     19.  CHKTRAP BFTST   ([8,SP])[0:4]  ;ESAMINA IL COD. OP.
          0008
00005408  6700 0006          20.          BEQ     CHK2SUB
0000540C  6000 0008          21.          BRA     CHKSUB
                             22.  *
                             23.  *       ROUTINE FITTIZIE
                             24.  *
00005410  4E71               25.  CHK2SUB NOP                    ;ELABORA COME RICHIESTO
                             26.  *
                             27.  *
00005412  4E4F               28.          TRAP    #15
00005414  0063               29.          DC.W    .RETURN
                             30.  *
00005416  4E71               31.  CHKSUB  NOP
                             32.  *
                             33.  *
00005418  4E4F               34.          TRAP    #15
0000541A  0063               35.          DC.W    .RETURN
                             36.  *
0000541C                     37.          END
```

Fig. 11.8 *Esempio di gestione delle eccezioni CHK e CHK2.*

11.2.4 Le istruzioni TRAPcc e TRAPV

L'istruzione TRAPcc è progettata per far scattare una trappola usando il vettore 7 (indirizzo $1C) della tabella di vettori di eccezione quando la condizione specificata "cc" è vera. Come per l'istruzione DBcc, ci sono 16 condizioni logiche possibili. Per contro, un'istruzione TRAPV fa scattare una trappola solamente quando il bit del codice di condizione di overflow è attivato (V = {1}). L'istruzione TRAPcc equivalente è TRAPVS. Queste due istruzioni (TRAPV e TRAPVS) svolgono la medesima funzione, ma non generano il medesimo codice di linguaggio-macchina definito nell'app. D. Inoltre, il vettore di eccezione 7 è condiviso dalle istruzioni TRAPcc, TRAPV, e da cpTRAPcc (dove "cp" sta per *Condition Coprocessor:* condizione di coprocessore).

L'istruzione TRAPcc. Come mostrato nella Tab. 11.4(a), l'istruzione TRAPcc ha diverse forme nel linguaggio assembler, che corrispondono alla sintassi ammessa per l'assemblatore dell'MC68020 della Motorola. Le condizioni "cc" sono definite nella Tab. 11.4(b).

Per esempio, le istruzioni

```
TEQ          ; Trappola se Z = {1}
```

e

```
TRAPEQ
```

sono identiche per l'assemblatore della Motorola. Quindi si può usare l'abbreviazione mnemonica "T" in luogo di "TRAP" per definire l'istruzione.

Un'altra forma dell'istruzione TRAPcc consente al programmatore di definire un valore costante che segue l'istruzione TRAPcc nella memoria. La designazione per l'assemblatore della Motorola è TRAPcc o TPcc. Questa costante può essere usata per passare l'informazione dal programma che ha causato la trappola al sistema operativo. Ad esempio, l'istruzione di "trappola se meno"

```
TPMI.W     #$DE
```

causa una trappola se N = {1} quando viene eseguita. Se si presenta la trappola, il frame di stack contiene sei word, proprio come per l'istruzione CHK descritta in precedenza. L'indirizzo dell'istruzione di trappola è memorizzato nella locazione (SP) + 8. Nella locazione di word della memoria che segue quella dell'istruzione di trappola, si trova la costante $DE in questo esempio. Come si rileva dalla Tab. 11.4(a), la costante può essere un valore di word o di longword, come richiesto.

Tab. 11.4 *L'istruzione TRAPcc.*

(a) Sintassi dell'istruzione.

Sintassi	Operazione
Tcc o TRAPcc	Trappola IF "cc" = vero ELSE prossima istruzione
TPcc.<l> #<dato> o TRAPcc<l> #<dato>	Trappola se "cc" = vero ELSE istruzione successiva dopo "<dato>"

Nota: <l> = W o L. Se <l> = W, <dato> è un valore di 16 bit. Se <l> = L, <dato> è un valore di 32 bit.

(b) Condizioni "cc"

CC	*Carry Clear*	Nessun riporto	0100	$\overline{C}$
CS	*Carry Set*	Riporto	0101	C
EQ	*EQual*	Uguale	0111	Z
F	*False*	Falso (mai vero)	0001	0
GE	*Greater or Equal*	Maggiore o uguale	1100	$N \cdot V + \overline{N} \cdot \overline{V}$
GT	*Greater Than*	Maggiore di	1110	$N \cdot V \cdot \overline{Z} + \overline{N} \cdot \overline{V} \cdot \overline{Z}$
HI	*HIgh*	Alto	0010	$\overline{C} \cdot \overline{Z}$
LE	*Less or Equal*	Minore o uguale	1111	$Z + N \cdot \overline{V} + \overline{N} \cdot V$
LS	*Low or Same*	Basso o uguale	0011	$C + Z$
LT	*Less Than*	Minore di	1101	$N \cdot \overline{V} + \overline{N} \cdot V$
MI	*Minus*	Meno	1011	N
NE	*Not Equal*	Non uguale	0110	$\overline{Z}$
PL	*Plus*	Più	1010	$\overline{N}$
N T	*True*	Vero (sempre)	0000	1
VC	*oVerflow Clear*	Nessun overflow	1000	$\overline{V}$
VS	*oVerflow Set*	Overflow	1001	V

Nota: Per alcuni assemblatori, "CS" = "LO" e "CC" = HS".

L'istruzione TRAPV. L'istruzione TRAPV è inclusa nell'insieme di istruzioni dell'MC68020 per garantire la compatibilità con i programmi scritti per l'MC68000 a 16 bit. Nei programmi per MC68020 in cui non sia richiesta tale compatibilità, si dovrebbe usare l'istruzione equivalente TRAPVS (*TRAP if V is Set:* trappola se V è attivato). Poiché TRAPcc non esiste nell'insieme di istruzioni dell'MC68000, tali programmi, scritti per la famiglia di processori dell'MC68020, non saranno eseguiti correttamente dai membri della famiglia a 16 bit.

ESERCIZI

11.2.1

Se l'istruzione

```
TRAP      #1
```

è situata alla locazione $10004 e viene eseguita quando (SR) = $A000 e (SP) = $7FFE, s'illustri il contenuto dello stack di sistema prima e dopo l'esecuzione dell'istruzione TRAP. S'inizializzi il vettore di TRAP #1 in modo che la routine di trappola inizi dalla locazione $8000.

11.2.2

Si scriva un programma che inserisca valori di word nello stack di utente a partire dalla locazione $3830. Se la lunghezza massima dello stack è di 10 word, si usi l'istruzione CHK per riportare il controllo al programma di supervisore quando lo stack subisce un overflow.

11.2.3

Adottando come modello il programma dell'esempio 9.8, si scriva un segmento di codice per verificare i confini dei registri usando l'istruzione CHK2. Si progetti la routine di gestione dell'eccezione per riportare i codici di condizione nel registro D1 se scatta una trappola e poi restituire il controllo al monitor o al sistema operativo.

11.2.4

Si modifichi il programma dell'esempio 11.3 per indicare la causa di una trappola di CHK come segue:

(a) Il valore del registro è al di sotto del confine inferiore.
(b) Il valore del registro è al di sopra del confine superiore.

S'imposti un flag nel registro D1 se i confini sono superati.

11.2.5

Si scriva una routine per determinare la causa di un eccezione di vettore 7 (TRAPcc, TRAPV o cpTRAPcc). Il programma dovrebbe dapprima operare una distinzione fra le tre classi di istruzioni che utilizzano il vettore 7. Dopodiché, per l'istruzione TRAPcc, dovrebbe decodificare la word di operazione ed usare la condizione come selettore di indirizzi in una tabella di salto. Tali indirizzi dovrebbero contenere l'indirizzo iniziale per l'appropriata routine di gestione della trappola.

11.3 TRAPPOLA DI ISTRUZIONE NON IMPLEMENTATA, TRACCIA E BREAKPOINT

L'MC68020 dispone di varie eccezioni speciali il cui scopo è quello di assistere nel debugging un progettista di sistema o di programma. Di solito, tali eccezioni sono impiegate per lo sviluppo ed il test di qualsiasi componente hardware e software necessario per una particolare applicazione. Per l'emulazione hardware

o software, si rivela utile la trappola di istruzione non implementata.[3] L'eccezione di traccia e l'istruzione BKPT (*BreaKPoinT*) servono ad assistere il programmatore nei test e nel debugging dei programmi. I breakpoint di programma possono anche essere impostati mediante l'istruzione ILLEGAL dell'MC68020. Nel presente paragrafo sarà discusso ciascuno di questi ausilii al debugging.

11.3.1 Trappola di istruzione non implementata

La CPU MC68020 riconosce come istruzioni "non implementate" quelle i cui primi quattro bit [15:12] sono {1010} ($A) o {1111} ($F). Esse sono denominate istruzioni di linea A e di linea F, rispettivamente. Il primo tipo causa l'esecuzione della routine di gestione dell'eccezione definita dal vettore 10 all'indirizzo $28. Il suo impiego più frequente è nell'emulazione di un'istruzione speciale o perfino delle capacità hardware di un altro sistema di computer. I rimanenti 12 bit nella word di operazione dell'istruzione della linea A possono essere usati per selezionare varie opzioni quando viene eseguita la routine di gestione dell'eccezione. Questa routine potrebbe eseguire la manipolazione di stringhe, implementare un algoritmo di trasformata veloce di Fourier, o fornire qualsiasi altra funzione conforme al progetto della routine. Per un programmatore di applicazioni, le istruzioni della linea A appaiono come "macroistruzioni" addizionali, aggiunte all'insieme ordinario dell'MC68020. In un programma in linguaggio assembler, l'istruzione

```
DC.W      $AXXX
```

dove "XXX" è un arbitrario valore esadecimale di tre cifre, farebbe scattare una trappola col vettore 10 qualora la CPU tentasse di eseguire questa istruzione di linea A. Quando si presenta l'eccezione, viene creato un frame di stack di quattro word, in cui vengono memorizzati il contenuto del registro di stato, l'indirizzo dell'istruzione non implementata ed una word di formato. L'indirizzo è ottenuto da una istruzione della forma:

```
MOVEA.L   (2,SP),A1
```

che oltrepassa il contenuto del registro di stato salvato nello stack e carica in A1 il valore salvato per il PC. Tale valore può essere utilizzato per individuare l'istruzione non implementata, al fine di decodificare qualsiasi altro bit usato per selezionare le opzioni per la routine di gestione dell'eccezione. Impiegando l'indirizzamento indiretto di memoria, l'istruzione

```
MOVE.W    ([2,SP]),D1
```

trasferirebbe in D1 l'istruzione di linea A senza utilizzare i registri d'indirizzo.

[3] L'*emulazione* — come viene usata qui — significa l'esecuzione dei programmi di un computer "target" e l'eventuale gestione di eventi esterni (ingresso/uscita, interruzioni, ecc.) in maniera realistica. Come definito nel cap. 1, un programma *simulatore* esegue i programmi di un computer target.

Le istruzioni di linea F. L'impiego di un'istruzione di linea F è simile a quello delle istruzioni di linea A se il sistema basato sull'MC68020 non contiene un coprocessore o se la CPU non riconosce l'istruzione di linea F come una valida istruzione di coprocessore.[4] Nel primo caso, le istruzioni di linea F sono tipicamente impiegate per emulare un coprocessore che non è presente. Se la CPU incontra un'istruzione di linea F che non è una valida istruzione di coprocessore, allora causerà un'eccezione di istruzione non implementata. La routine di gestione dell'eccezione può eseguire qualsiasi funzione prevista dal suo progetto.

Quando un coprocessore è presente nel sistema, la CPU passerà l'istruzione di linea F al coprocessore, come descritto nel cap. 12. Tranne che per programmare un coprocessore o per emulare un coprocessore non presente, un programmatore non dovrebbe generalmente usare le istruzioni di linea F dell'MC68020 per altri scopi, poiché esse sono riservate per il controllo del coprocessore.

11.3.2 Eccezioni di traccia

Durante i test di un programma in fase di sviluppo, è spesso comodo per un programmatore poter far sì che la CPU esegua una singola istruzione o una breve sequenza di istruzioni, seguite da una visualizzazione di informazioni utili ai fini del debugging. L'MC68020 dispone di due eccezioni di traccia che soddisfano questi requisiti per i test dei programmi. Esse sono denominate traccia di *singola istruzione* e traccia su *cambiamento di flusso*, rispettivamente. Il tracciamento può avvenire sia nella modalità di supervisore che in quella di utente. Comunque, i bit di traccia (T0, T1) nel registro di stato possono essere modificati soltanto da un'istruzione eseguita nel modo di supervisore. Un sistema operativo o un programma di monitor possono abilitare la traccia di singola istruzione mediante l'istruzione

```
ORI        #$8000,SR        ; pone T1 = {1}
```

come descritto nel par. 10.1. La traccia sul cambiamento di flusso potrebbe essere abilitata dall'istruzione

```
ORI        $4000,SR  ; pone T0 = {1}
```

come pure da altre istruzioni descritte nel cap. 10. Queste modalità di traccia sono disabilitate azzerando T0 e T1. Quando l'eccezione di traccia viene riconosciuta dalla CPU dopo che è stata eseguita l'istruzione che ha causato la traccia, viene eseguita la routine di gestione dell'eccezione di traccia, usando il vettore 9 all'indirizzo $024. Nella routine suddetta, il tracciamento viene disabilitato dalla CPU, che crea un frame di stack di quattro word prrima che il controllo sia trasferito alla routine. Il valore del PC nello stack all'indirizzo (SP) + 2 è l'indirizzo della successiva istruzione da eseguire, se viene eseguita un'istruzione RTE nella routine di traccia.

[4] Se una valida istruzione di coprocessore viene eseguita in un sistema basato sull'MC68020 senza un coprocessore, la CPU dovrebbe ricevere un segnale di errore di bus dalla circuiteria esterna quando la CPU tenta di accedere al coprocessore assente. Ciò è causa di un'eccezione di istruzione non implementata.

A meno che il contenuto del registro di stato all'indirizzo (SP) nello stack non venga modificato dalla routine di traccia per disabilitare il tracciamento, questo sarà abilitato di nuovo dopo l'esecuzione di RTE.

Quando T1 = {1} nel registro di stato, l'eccezione di traccia viene generata ogni volta che un'istruzione qualsiasi è stata eseguita. Se c'è un'interruzione in sospeso, l'eccezione di traccia viene elaborata prima dell'eccezione d'interruzione. Quindi viene creato dapprima il frame di stack per la traccia, seguito dal frame di stack per l'interruzione. Comunque, la routine di gestione dell'eccezione d'interruzione è la prima ad essere eseguita. Queste priorità per l'elaborazione delle eccezioni saranno descritte ulteriormente nel par. 11.7. Se il tracciamento è abilitato quando viene eseguita un'istruzione illegale o non implementata, l'eccezione di traccia non sarà attivata finché l'istruzione non sarà stata eseguita.

L'eccezione di traccia sul cambiamento di flusso (T0 = {1}) viene attivata quando è stata abilitata da una delle seguenti condizioni:

(a) Un'istruzione causa l'aggiornamento non sequenziale del contatore di programma.

(b) Un'istruzione modifica il registro di stato.

(c) Un'istruzione di coprocessore causa l'incremento non sequenziale del contatore di programma.

Le istruzioni e le condizioni che causeranno una traccia sul cambiamento di flusso includono tutte le istruzioni di salto di tipo "branch" allorché il salto viene intrapreso, istruzioni di salto di tipo "jump", trappole, e istruzioni di ritorno da subroutine o da routine di gestione delle eccezioni. Anche certe istruzioni di coprocessore ed alcune che modificano il contenuto del registro di stato possono causare un'eccezione di traccia sul cambiamento di flusso quando è abilitata questa modalità di tracciamento.

La tipica routine di traccia è progettata per visualizzare i contenuti dei registri della CPU e delle locazioni di memoria associate con l'esecuzione del programma in tracciamento. Alcuni esempi della funzione di traccia del monitor 133BUG sono stati forniti nel cap. 5. Per tale monitor, le modalità di traccia sono avviate dal comando T (traccia di un'istruzione) o TC (traccia sul cambiamento di flusso) come comandi dell'operatore. Si potrebbe scrivere una routine speciale di traccia per svolgere qualunque funzione si desideri, qualora il tracciamento standard fornito dal monitor non fosse sufficiente per un'applicazione.

11.3.3 Breakpoint e l'istruzione BKPT

Le opzioni di traccia per l'MC68020 consentono di sospendere l'esecuzione normale del programma quando si presentano certi eventi. In particolare, un'eccezione di traccia potrebbe essere generata dopo l'esecuzione di una singola istruzione se SR[15] = {1} o dopo che un'istruzione ha causato un cambiamento del

flusso di controllo se SR[14] = {1}. In molti casi, un programmatore o un progettista dell'hardware desidera sospendere la normale esecuzione del programma quando la CPU indirizza una certa locazione nella memoria. Come avviene per le eccezioni di traccia, il controllo dovrebbe essere passato ad una routine per assistere nel debugging allorché viene effettuato l'accesso designato. Se la locazione contiene un'istruzione, si dice che il programma ha raggiunto un *breakpoint*. Di solito, in tale punto viene eseguita una routine di gestione dell'eccezione, che visualizza i contenuti dei registri ed altre informazioni d'interesse concernenti il programma in esame.

Nei sistemi basati sull'MC68020 che non dispongono di un'unità di gestione della memoria MC68851, ci sono vari modi per definire i breakpoint in un programma. Il primo consiste nell'impiego dell'istruzione

ILLEGAL

con la word di operazione $4AFC. Quando viene incontrata, essa causa un'eccezione di istruzione illegale, che usa il vettore 4 alla locazione $010. Un'altro metodo è quello d'impiegare una delle istruzioni TRAP #<N>. Sia ILLEGAL che TRAP causeranno l'elaborazione dell'eccezione, quando compaiono nel flusso di istruzioni di un programma in fase di debugging. Normalmente, su comando dell'operatore, il programma di debugging del software di sistema sostituisce un'istruzione alla locazione designata con una che causa un'eccezione. Nei sistemi basati sull'MC68020 senza un MC68851, si utilizza l'istruzione ILLEGAL o una delle istruzioni TRAP per sostituire l'istruzione nel breakpoint. Quindi, l'una o l'altra istruzione all'indirizzo di breakpoint causa un'eccezione allorché il programma viene eseguito. Dopo che la routine di gestione dell'eccezione ha espletato la propria funzione di ausilio al debugging del programma, l'istruzione ILLEGAL o TRAP nella routine dev'essere sostituita da quella originale, se il programma dovrà proseguire l'esecuzione dalla locazione di brekpoint.

L'MC68020 ha un'istruzione di breakpoint (BKPT) che richiede l'impiego di una circuiteria esterna per fissare e rimuovere i breakpoint da un programma in fase di test. L'uso più frequente di questa istruzione si ha quando il sistema del computer dispone di un'unità di gestione della memoria MC68851, poiché tale unità è in grado di svolgere le operazioni necessarie al supporto dell'istruzione BKPT. Per ulteriori dettagli, il lettore può consultare l'*MC68851 User's Manual*.

ESERCIZI

11.3.1 Usando le istruzioni di linea A, si emuli una "macroistruzione" che trasferisce una stringa di caratteri di lunghezza arbitraria da un'area della memoria ad un'altra. Si supponga che D1 contenga la lunghezza, A1 l'indirizzo iniziale della stringa e A2 la locazione di destinazione per il primo carattere da trasferire.

11.3.2 Qual è lo svantaggio per una routine di traccia che si limita a visualizzare i valori in ogni registro della CPU dopo l'esecuzione di ciascuna istruzione?

11.3.3 Si supponga che il tracciamento sia abilitato quando si presentano le seguenti condizioni:

(a) STOP #$2000 viene eseguita.
(b) La CPU incontra un'istruzione illegale in un flusso di istruzioni.

Qual è il comportamento della CPU in questi casi?

11.3.4 Si scriva una semplice routine di traccia per stampare i contenuti di tutti i registri del processore quando viene attivata un'eccezione di traccia. Si verifichi il tracciamento scrivendo un semplice programma in cui viene tracciata ogni istruzione. (*Nota:* La routine per la stampa dei valori è fornita dal sistema operativo o dal programma di monitor nella maggior parte dei sistemi. Questo impiego del monitor 133BUG per convertire i valori in ASCII e per visualizzare i risultati è stato discusso nei capitoli dal 5 al 7.)

11.3.5 Si definiscano le caratteristiche desiderabili di una routine di debugging generalizzata per assistere un programmatore durante il test e il debugging dei programmi.

11.4 ERRORI DI PROGRAMMA CHE CAUSANO TRAPPOLE

L'MC68020 è progettato per proteggere il sistema di computer da errori che potrebbero dar luogo ad un comportamento imprevedibile. La Tab. 11.5 elenca gli errori di programma che sono "intrappolati" dalla CPU. La violazione di privilegio, l'istruzione illegale e le trappole di errori di indirizzo di solito si presentano durante il debugging di un programma. Un sistema operativo o un programma di monitor causeranno di solito la terminazione prematura ("aborto") del programma che ha causato l'eccezione. Nella maggior parte dei casi, non è desiderabile il proseguimento dell'esecuzione di un programma che ha causato un simile evento, tranne forse in circostanze speciali che dipendono interamente dai requisiti dell'applicazione.

11.4.1 Violazione di privilegio

Se un programma che opera nella modalità di utente tenta di eseguire una delle istruzioni privilegiate elencate nella Tab. 11.5, viene causata un'eccezione mediante il vettore 8 alla locazione $020. Viene creato un frame di stack di quattro word ed il valore del PC indica la locazione dell'istruzione che ha causato la violazione.

Tab. 11.5 *Errori di programma che causano trappole.*

Errore	Causa	Commenti
Violazione di privilegio	In modo utente; tentativo di eseguire un'istruzione privilegiata.	Se S = {0}, tentativo di eseguire: ANDI, EORI, MOVE o ORI a SR; MOVE da SR; MOVE a USP; MOVEC; MOVES; RESET; RTE; STOP; cpRESTORE; cpSAVE.
Istruzione illegale	Configurazione di bit del codice operativo non riconosciuta.	Il valore di PC sullo stack è l'indirizzo dell'istruzione illegale.
Errore di indirizzo	Tentativo di prelievo del l'istruzione a un indirizzo dispari.	Viene creato un frame di stack di eccezione di errore di bus.

Note:
1. Quando si presentano queste eccezioni, l'istruzione che ha causato l'eccezione non viene eseguita.
2. L'eccezione di istruzione illegale può essere usata per implementare una caratteristica di breakpoint, come descritto nel par. 11.3.

11.4.2 Istruzione illegale

La trappola di *istruzione illegale*, con vettore alla locazione $10, è usata per proteggere il sistema dagli effetti di un codice di macchina scorretto o da un malfunzionamento locale della memoria. Se la CPU non riconosce la configurazione di bit di un'istruzione come valida, allora viene generata un'eccezione di istruzione illegale.[5] L'eccezione può essere causata deliberatamente se viene inclusa l'istruzione ILLEGAL in un programma.

11.4.3 Errore di indirizzo

Se il processore tenta di accedere ad un'istruzione ad un indirizzo dispari, viene causata un'eccezione di errore di indirizzo mediante il vettore alla locazione $0C. L'informazione salvata nello stack può essere utilizzata per diagnosticare il problema. Il frame di stack creato è simile a quello di un'eccezione di errore di bus corto, che sarà descritta nel par. 11.7.

---

[5] Questa eccezione viene generata anche se si tenta di eseguire un'istruzione MOVEC senza una corretta specificazione di registro.

ESERCIZI

11.4.1 Si consideri l'impiego dell'eccezione di violazione di privilegio allorché un nuovo sistema operativo è in fase di sviluppo. Il nuovo sistema operativo dovrebbe essere in grado di eseguire istruzioni privilegiate. Tuttavia, a causa del progetto dell'MC68020, il sistema operativo in fase di sviluppo deve operare nel modo di utente. Inoltre, le eccezioni devono essere gestite dal sistema operativo effettivo per il computer di sviluppo (cioè, il sistema operativo che opera nel modo di supervisore). Si supponga che il nuovo sistema operativo abbia la propria tabella di vettori. Si spieghi il modo in cui il sistema operativo effettivo potrebbe consentire al sistema operativo in fase di sviluppo di emulare istruzioni privilegiate.

11.4.2 Si scriva una routine di errore d'indirizzo per decodificare l'informazione salvata nello stack ed inviarla in uscita al terminale dell'operatore. Si provi la routine provocando un errore d'indirizzamento e visualizzando i valori salvati di (SR) e (PC).

11.4.3 Si descrivano alcuni modi in cui potrebbe presentarsi una trappola d'istruzione illegale durante l'esecuzione di un programma. Si considerino errori sia nell'hardware che nel software che potrebbero essere prodotti dalla creazione di un'istruzione illegale.

11.5 ERRORI E CONDIZIONI DI SISTEMA

La CPU riconosce un certo numero di errori e condizioni di sistema che servono a controllare e proteggere il sistema del computer. Una delle più importanti in sistema basati sull'MC68020 è l'eccezione di errore di bus causata dalla circuiteria esterna. Come sarà descritto nella prima parte di questo paragrafo, l'errore di bus è impiegato per una varietà di scopi, tra cui la rivelazione degli errori in un sistema di computer. In questo paragrafo saranno trattate anche le eccezioni di errore di formato, lo stato di arresto e le eccezioni generate dal coprocessore.

11.5.1 L'eccezione di errore di bus

Un'eccezione di errore di bus è causata dalla linea di segnale di errore di bus dell'MC68020. Questo segnale può essere attivato dalla circuiteria esterna speciale o da un'unità di gestione della memoria impaginata (*Paged Memory Management Unit:* PMMU) MC68851 se tale unità è presente nel sistema. Le cause di un errore di bus non sono "standard", tranne che per l'MC68851. Pertanto l'MC68020 deve determinare l'azione da intraprendere conformemente al progetto della routine di eccezione dell'errore di bus. La Tab. 11.6 elenca le possibili situazioni tipiche che possono causare l'attivazione della linea di segnale di errore di bus in un sistema di computer basato sull'MC68020.

Tab. 11.6 *Esempio d'impiego del segnale di errore di bus.*

Situazione	Causa dell'errore di bus	Azione della CPU
Nessuna risposta di una locazione di memoria o di un dispositivo periferico	Timer "cane da guardia" (*watchdog*)	Eccezione di errore di bus
Errore di memoria	Circuiteria di correzione dell'errore	Eccezione di errore di bus
Errori o condizioni rivelate dall'MC68851	Unità di gestione della memoria MC68851	Eccezione di errore di bus
Violazione di privilegio		
Difetto di pagina in un sistema di memoria virtuale		
Tentativo di scrittura su una pagina protetta		
Violazione del livello di accesso (CALLM)		
Nessun coprocessore presente (emulazione di coprocessore)	Timer "cane da guardia"	Eccezione di emulatore di linea F
Errore durante l'esecuzione di un'istruzione di coprocessore	Circuiteria speciale	Eccezione di errore di bus
Ciclo di riconoscimento di breakpoint	MC68851	Eccezione di istruzione illegale nel breakpoint
Ciclo di riconoscimento dell'interruzione	Circuiteria speciale	Eccezione di interruzione spuria

Nota: Le linee di segnale dell'MC68020 saranno discusse in dettaglio nel par. 13.4.

In genere, la linea di segnale di errore di bus viene attivata da una circuiteria esterna alla CPU quando si verifica un'errore o una condizione speciale. La linea di segnale di errore di bus indica che un dispositivo esterno non è in grado di completare l'istruzione della CPU in corso di esecuzione. Quando la CPU esegue un'istruzione, essa attende una risposta da un dispositivo esterno selezionato per indicare che il dispositivo riconosce l'istruzione e può completare l'elaborazione come richiesto.[6] Se il dispositivo non può rispondere correttamente o se un errore viene rivelato nell'istruzione, il dispositivo dovrebbe attivare la linea di segnale di errore di bus.

11.5.2 L'errore di formato

Quando viene eseguita l'istruzione RTE, viene controllata la validità del codice di formato alla locazione (SP) + 6 nello stack. Se tale codice non è corretto, viene generata un'eccezione di errore di formato. Se la RTE viene eseguita dopo che un'eccezione di errore di bus è stata riconosciuta, viene verificata la validità anche degli altri dati per il processore memorizzati nello stack, prima che tali informazioni (16 word o 46 word) siano ripristinate nei registri interni della CPU. Se questi valori non fossero corretti, sarebbe generata un'eccezione di errore di formato. In ogni caso, il verificarsi di un errore di formato implica di solito che il contenuto dello stack o del puntatore di stack è stato alterato in una maniera non valida tra l'istante in cui l'eccezione è stata riconosciuta ed il momento in cui viene eseguita l'istruzione RTE.

11.5.3 Stato di arresto

Come descritto nel cap. 10, lo stato di arresto ("*halt*") viene attivato allorché la CPU determina di non essere più in grado di continuare ad elaborare le istruzioni. Si entra in questo stato solamente quando viene rivelato un malfunzionamento irrimediabile nel sistema. Una volta che è entrato nello stato di arresto, il processore potrà essere riavviato soltanto da un reset da parte della circuiteria esterna.

11.5.4 Eccezioni di coprocessore

Certe eccezioni sono rivelate da un coprocessore della famiglia dell'MC68020 ed indicate alla CPU. Le violazioni di protocollo rivelate dal coprocessore fanno sì che la CPU intraprenda un'eccezione mediante il vettore 13. Le istruzioni di trappola del coprocessore (cpTRAPcc) causano un'eccezione che usa il vettore 7.

Altri vettori di eccezione relativi al coprocessore, mostrati nella Fig. 11.2, sono definiti per coprocessori specifici. Il coprocessore in virgola mobile MC68881

[6] Le linee di segnale interessate nel ciclo di bus della CPU sono definite nel par. 13.4.

definisce le eccezioni causate da errori di elaborazione di dati o da altre condizioni. Queste eccezioni sono elaborate dalla CPU mediante i vettori dal 48 al 54. Similmente, all'unità di gestione della memoria MC68851 sono assegnati i vettori da 56 al 58 per l'elaborazione delle eccezioni da parte della CPU delle condizioni relative all'MC68851 che causano un'eccezione. Altre eccezioni relative al coprocessore saranno descritte più dettagliatamente nel cap. 12, in cui saranno trattati i coprocessori. L'azione della routine di gestione delle eccezioni per un'eccezione di coprocessore dipende interamente dal progetto del software di sistema per il computer.

ESERCIZIO

11.5.1

Si supponga che un indirizzo illegale (dispari) sia memorizzato nella locazione del vettore di errore d'indirizzo. Che risultato si otterrebbe se il processore rivelasse un errore d'indirizzo durante l'esecuzione di un programma?

11.6 GESTIONE DELLE INTERRUZIONI DA PARTE DELL'MC68020

Il sistema di interruzioni dell'MC68020 consente ad un dispositivo esterno di interrompere l'esecuzione del processore e causa il trasferimento del controllo del programma ad una routine di gestione dell'interruzione. Questo paragrafo riguarda gli aspetti di programmazione della sequenza d'interruzione dell'MC68020, inclusi il metodo di priorità per le interruzioni e l'attività del processore durante l'elaborazione delle interruzioni. Le linee di segnale usate per le richieste ed i riconoscimenti delle interruzioni saranno discusse nel cap. 13.

Una richiesta d'interruzione da parte di un dispositivo esterno può avvenire in uno di sette livelli di priorità, in base al valore presente sulle tre linee di segnale d'interruzione del processore. Alle interruzioni sono assegnate delle priorità, dal livello minimo (1) a quello massimo (7). Tali priorità permettono che una routine che sta elaborando un'interruzione ad un certo livello possa essere interrotta da una richiesta d'interruzione di livello superiore. Al termine dell'elaborazione dell'interuzione di livello superiore, il controllo ritorna alla routine d'interruzione di livello inferiore, che era rimasta in attesa di completare la sua esecuzione. Una volta che tutte le richieste d'interruzione sono state elaborate, il controllo torna infine al programma che era stato interrotto.

Dal punto di vista del processore, un'interruzione è una richiesta generata esternamente per l'elaborazione delle eccezioni. La richiesta d'interruzione può essere considerata attiva, in sospeso, o disabilitata. Una richiesta *attiva* viene elaborata immediatamente dopo il completamento di qualunque istruzione in corso di esecuzione, purché nessuna eccezione di priorità superiore acquisisca la precedenza. La richiesta è *in sospeso* se al momento il processore sta elaborando una eccezione di priorità superiore. Le richieste in sospeso potranno essere servite al

termine dell'elaborazione di priorità superiore, a meno che il processore non entri in uno stato di arresto. Se un livello d'interruzione è disabilitato, una richiesta d'interruzione a tale livello sarà ignorata finché il livello non verrà abilitato modificando la maschera d'interruzione nel registro di stato, cioè i bit (SR)[10:8].

Il sistema di interruzioni è inizializzato da un programma supervisore durante l'inizializzazione del sistema, caricando nelle locazioni appropriate della tabella di vettori gli indirizzi iniziali di ciascuna routine d'interruzione da utilizzare. Tale inizializzazione viene effettuata con tutti i livelli d'interruzione disabilitati, tranne il livello 7, che non può essere disabilitato. I livelli d'interruzione vengono abilitati appena prima che il controllo venga passato dal programma di supervisore al primo programma applicativo da eseguire.

I bit di maschera d'interruzione sono mostrati nella Fig. 11.9, che include il valore di maschera per ciascun livello. In generale, quando la richiesta d'interruzione è al livello mascherato o a quello inferiore, la richiesta d'interruzione non sarà accettata. La richiesta di livello 7 è un'eccezione a questa regola e sarà elaborata indipendentemente dall'impostazione della maschera d'interruzione. Se la maschera è impostata a {000}, tutti i livelli d'interruzione sono abilitati.

Registro di stato

15	14		10	9	8		1	0 ← Numero di bit
		...	I_2	I_1	I_0	...		

Livelli disabilitati (mascherati)	Maschera d'interruz.		
	I_2	I_1	I_0
1-6	1	1	1
1-6	1	1	0
1-5	1	0	1
1-4	1	0	0
1-3	0	1	1
1-2	0	1	0
1	0	0	1
Tutti i livelli abilitati	0	0	0

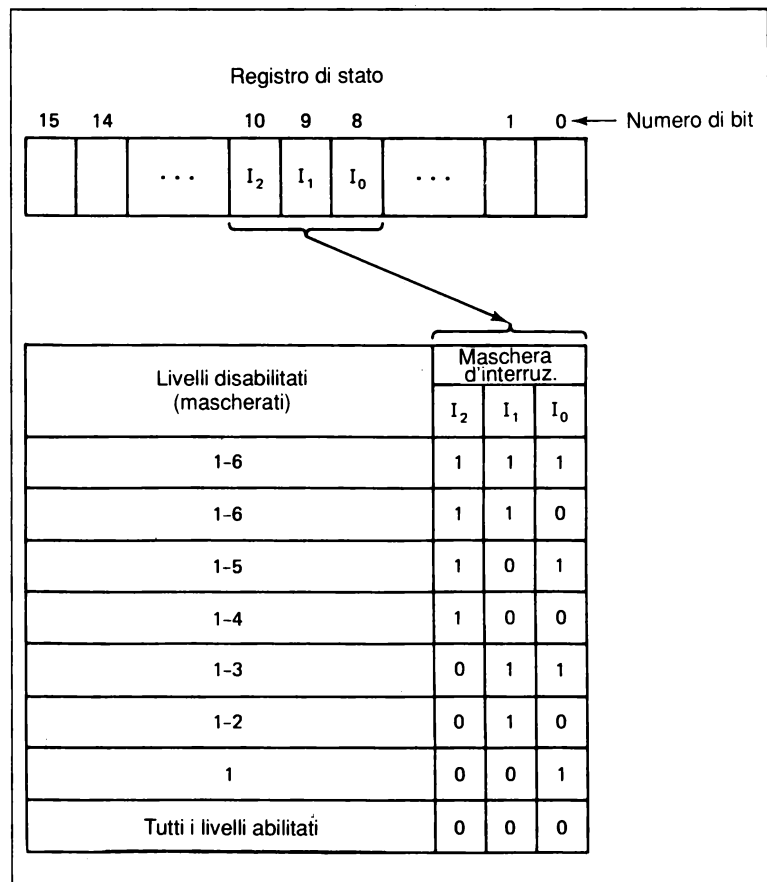



Fig. 11.9 *Maschera d'interruzione per l'MC68020.*

La tabella di vettori per le interruzioni è mostrata nella Tab. 11.7. Il vettore d'interruzione spuria viene usato quando il processore riconosce una richiesta d'interruzione ma esiste esternamente qualche condizione di errore. Un errore di bus da un dispositivo esterno durante una richiesta d'interruzione causerà un'interruzione spuria. Altrimenti, l'indirizzo iniziale della routine d'interruzione per i livelli da 1 a 7 sarà prelevato dall'appropriata locazione di vettore. Le interruzioni sono classificabili in autovettori ed interruzioni di utente. La selezione tra queste due modalità operative per il sistema d'interruzione dipende interamente dalla circuiteria esterna. In entrambi i casi, l'indirizzo dell'interruzione viene calcolato come il quadruplo del numero del vettore. La differenza nel funzionamento dell'hardware non ha alcun effetto sul progetto delle routine d'interruzione.

Tab. 11.7 *Esempio d'impiego del segnale di errore di bus.*

Numero del vettore (decimale)	**Locazione di memoria (esadecimale)**	**Nome**
15	$003C	Vettore d'interruzione non inizializzata
24	$0060	Vettore d'interruzione spuria
25	$0064	Autovettore di livello 1
26	$0068	Autovettore di livello 2
27	$006C	Autovettore di livello 3
28	$0070	Autovettore di livello 4
29	$0074	Autovettore di livello 5
30	$0078	Autovettore di livello 6
31	$007C	Autovettore di livello 7
.	.	.
.	.	.
.	.	.
64	$0100	Vettore 1 d'interruzione di utente
65	$0104	Vettore 2 d'interruzione di utente
.	.	.
.	.	.
.	.	.
255	$03FC	Vettore 192 d'interruzione di utente

Note:
1. Il vettore 15 dovrebbe essere fornito da un dispositivo esterno non inizializzato se la CPU richiede un numero di vettore.
2. Un'interruzione spuria ha luogo quando la CPU rivela un errore durante l'elaborazione dell'interruzione.

Fig. 11.10
Elaborazione di un'interruzione tramite lo stack d'interruzione.

Richiesta di interruzione

Esamina i bit d'interruzione in SR

Livello richiesto abilitato?

NO

Continua l'elaborazione

SI

Salva (SR) internamente
Genera un nuovo (SR):
- Modo di supervisore
- Traccia disabilitata
- Livello richiesto

Calcola l'indir. del vettore
Indirizzo = 4 * num. vettore

Inserisce nello stack di supervisore: la word di formato, (PC), (SR)

(PC) ← (indirizzo vettore)

Routine di interruzione

L'MC68020 ha due puntatori di stack di supervisore, designati come puntatore di stack di interruzione (*Interrupt Stack Pointer:* ISP) e puntatore di stack principale (*Master Stack Pointer:* MSP). Lo stack d'interruzione indirizzato da ISP in un programma in linguaggio assembler è lo stack di supervisore attivo per tutte le eccezioni quando il bit M del registro di stato vale {0}: cioè, (SR)[12] = {0}. L'elabora-

zione delle interruzioni mediante lo stack d'interruzione è descritta all'inizio del paragrafo. Quando il sistema operativo attiva lo stack principale ponendo M = {1} nel registro di stato, lo stack principale viene utilizzato per elaborare tutte le eccezioni. Comunque, anche lo stack d'interruzione è utilizzato per elaborare le interruzioni, come descritto in questo paragrafo. Sono quindi disponibili due puntatori di stack quando lo stack principale è impiegato da routine del sistema operativo. Tuttavia, un programma può agire su un solo stack alla volta. La CPU determina quale stack è attivo quando un programma fa riferimento al puntatore di stack di supervisore.

11.6.1 Elaborazione delle interruzioni mediante lo stack d'interruzione

Se il processore esegue le istruzioni nello stato normale, una richiesta d'interruzione che viene riconosciuta e diviene attiva genera una sequenza di eventi concepiti per passare il controllo ad una routine d'interruzione prestabilita. Questa routine elabora l'interruzione come richiesto, dopodiché restituisce il controllo al programma interrotto. La sequenza di eventi è mostrata nella Fig. 11.10. Le operazioni sono eseguite dall'hardware finché il controllo non viene passato alla routine d'interruzione. L'elaborazione richiesta nella routine d'interruzione dipende interamente dall'applicazione. L'impiego di interruzioni per la programmazione di I/O sarà considerato nel par. 13.2.

Esempio 11-4

L'elaborazione di un'interruzione consiste nel salvataggio della word di formato, di (PC) e di (SR) nello stack di supervisore, se l'interruzione viene riconosciuta mentre viene eseguita un'istruzione dell'MC68020 (viene creato uno stack di 10 word se nessuna interruzione viene riconosciuta durante l'esecuzione di un'istruzione di coprocessore). Se il bit M di stack principale nel registro di stato vale {0}, il puntatore di stack di supervisore è il puntatore di stack d'interruzione, designato come ISP.

Come esempio, si consideri la Fig. 11.11 e si supponga che un'interruzione si presenti quando i contenuti iniziali dei registri selezionati hanno i seguenti valori:

(SR) = $2008
(PC) = $0000164A
(ISP) = $0000FFFE

indicanti che un programma in modo supervisore è in procinto di eseguire l'istruzione alla locazione $164A allorché l'interruzione viene riconosciuta. L'elaborazione produce l'inserimento nella locazione $FFFC di una word di formato con codice di formato $0 ed un offset di vettore $74, seguita da (PC) nella locazione di longword $FFF8, e da (SR) nella locazione $FFF6.

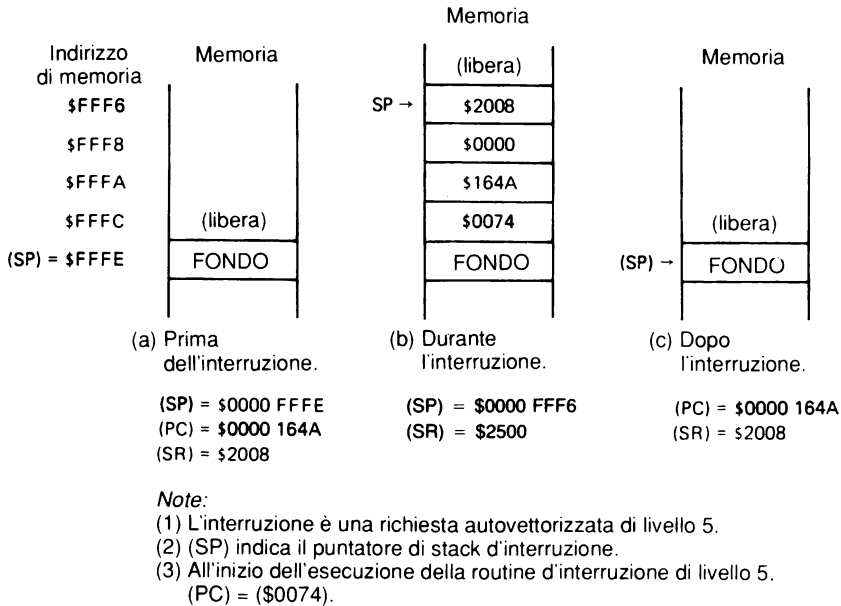



Fig. 11.11 *Lo stack di sistema durante l'elaborazione dell'interruzione.*

La routine d'interruzione può usare lo stack di supervisore come richiesto, purché ISP sia ripristinato al valore $0000FFF6 prima che la routine esegua un'istruzione RTE e restituisca il controllo al programma interrotto. Una routine di gestione dell'eccezione per l'interruzione autovettorizzata di livello 5 è individuata dall'indirizzo (vettore 29) memorizzato nella locazione $0074.

La routine di gestione dell'interruzione inizia l'esecuzione con:

(PC) = ($0074)
(SR) = $2500

che consente l'esecuzione alla locazione vettorizzata con le interruzioni di livello 5 e inferiori disabilitate. Quando viene eseguita l'istruzione RTE nella routine di gestione dell'interruzione, il controllo viene restituito al programma interrotto nella locazione $000164A.

Esempio 11-5

Le linee di segnale esterne determinano se la modalità d'interruzione è di *autovettore* o di *interruzione di utente*. Quest'ultima è denominata anche richiesta d'interruzione vettorizzata. Quando il modo di autovettore viene richiesto da un dispositivo esterno, la CPU fornisce automaticamente la locazione del vettore. Altrimenti, nel modo (vettorizzato) di interruzione di utente, il numero del vettore dev'essere fornito dal dispositivo esterno. Questo numero viene moltiplicato per 4 per ottenere l'indirizzo del vettore di eccezione. Il modo in cui il numero del vettore è fornito alla CPU dal dispositivo esterno sarà discusso nel par. 13.4.

In entrambi i casi, il programmatore deve conoscere le seguenti informazioni per programmare una routine d'interruzione:

(a) Il numero del vettore (o indirizzo).

(b) La priorità assegnata dal progetto hardware del sistema per il mascheramento o lo smascheramento di interruzioni vettorizzate (la priorità per autovettori è fissata).

(c) Dettagli delle operazioni funzionali della routine d'interruzione, come stabilite dai requisiti dell'hardware.

L'ubicazione della routine d'interruzione viene decisa solitamente durante la fase di progettazione del software; la priorità di ciascuna interruzione è determinata dai requisiti del sistema, in primo luogo dai vincoli di tempo dei dispositivi esterni. La priorità d'interruzione è importante per il programmatore soltanto se la routine d'interruzione gestisce la maschera d'interruzione nel registro di stato modificando (SR)[10:8].

11.6.2 Il puntatore di stack d'interruzione ed il puntatore di stack principale

Un sistema operativo per un sistema basato sull'MC68020 potrebbe utilizzare lo stack di sistema per elaborare le eccezioni. Quando M = {1} nel registro di stato, qualsiasi eccezione causa la creazione dell'appropriato frame nello stack principale. Il valore del bit M non influsce sul modo di privilegio della CPU, ma l'impiego dello stack principale permette ad un sistema operativo di separare dalle interruzioni le eccezioni relative al programma. Infatti, in un sistema operativo multiprogrammato, ciascun task (compito) potrebbe avere il proprio spazio di stack di supervisore poiché il sistema operativo potrebbe modificare appropriatamente il valore di (MSP) prima che il controllo sia passato ad un task specifico. L'area di

stack per l'elaborazione di un'interruzione individuata da (ISP) dovrebbe essere tenuta separata nella memoria da ciascuno degli stack relativi al task.

Dal momento che la differenza tra lo stack principale e lo stack d'interruzione è d'interesse soprattutto per quei programmatori di sistema che creano sistemi operativi, il lettore interessato potrà reperire ulteriori dettagli nell'*MC68020 32-Bit Microprocessor User's Manual*. Anche quando il puntatore di stack principale è il puntatore di stack di supervisore, le interruzioni vengono elaborare usando lo stack d'interruzione, come descritto nel sottopar. 11.6.1. L'impiego dello stack principale con sistemi operativi multiprogrammati sarà descritto nel par. 12.5.

ESERCIZI

11.6.1 Quali sono alcune applicazioni di sistema per l'interruzione non mascherabile di livello 7?

11.6.2 Quanti vettori d'interruzione, tra vettorizzati ed autovettorizzati, sono disponibili in base alla tabella dei vettori di eccezione? Il numero del vettore è un intero di 8 bit che consente 256 entrate per indirizzare la tabella dei vettori di eccezione, ma — stando alla tabella — non tutte le entrate possono essere associate con interruzioni. Oppure potrebbero esserlo? In altre parole, un progettista è davvero impossibilitato ad usare i vettori da 0 a 23 o i vettori da 32 a 47 come vettori d'interruzione?

11.7 FRAME DI STACK E PRIORITA' DELLE ECCEZIONi

In questo paragrafo, viene riepilogato un certo numero di dettagli concernenti l'elaborazione delle eccezioni. Dapprima viene presentato il formato del frame di stack per ciascuna eccezione.[7] Dopodiché, viene definita la priorità delle eccezioni per determinare l'eccezione che dev'essere elaborata per prima allorché due eccezioni si presentano simultaneamente.

11.7.1 Frame di stack

Il frame di stack che viene creato quando un'eccezione viene riconosciuta può contenere da un minimo di 4 word ad un massimo di 46 word. La maggior parte delle eccezioni creano frame di stack di 4 o di 6 word. Un'eccezione di errore di bus può creare un frame di stack di 16 word o di 46 word, a seconda dell'istante in cui avviene l'eccezione nel ciclo di istruzione della CPU.

[7] Il frame di stack è stato presentato nel par. 9.4. Esso è stato usato per la memorizzazione temporanea di variabili durante la chiamata ad una subroutine. I frame di stack di eccezione sono creati automaticamente nello stack allorché viene riconosciuta un'eccezione.

Frame di stack di quattro word. La Fig. 11.3 ha illustrato il frame di stack fondamentale di 4 word creato dall'elaborazione di un'eccezione. Questo frame di stack è per interruzioni, errori di formato, istruzioni TRAP, istruzioni illegali, trappole di emulatore di linea A e di linea F, violazioni di privilegio, e certe eccezioni di coprocessore.

Frame di stack di sei word. Il frame di stack di sei word della Fig. 11.3 contiene l'indirizzo dell'istruzione che ha causato l'eccezione, come pure il formato della word, i valori di (PC) e di (SR). Se viene eseguita un'istruzione RTE che utilizza questo frame di stack, il (PC) indica l'istruzione successiva da eseguire. Questo frame di stack è creato da CHK, CHK2, cpTRAPcc, TRAPcc, TRAPV, traccia, divisione per zero, nonché da certe istruzioni di coprocessore.

Frame di stack di difetto di ciclo di bus corto (16 word). Se si verifica un errore di bus quando la CPU sta avviando l'esecuzione di un'istruzione, viene creato il frame di stack di 16 word mostrato nella Fig. 11.12. Le informazioni vengono ripristinate nei vari registri della CPU se l'istruzione RTE viene eseguita dopo che la routine di gestione dell'eccezione ha corretto la situazione che ha causato l'eccezione.

	15	0
SP →	Registro di stato	
+ $02	Contatore di programma	
+ $04		
+ $06	1 0 1 0	Offset del vettore
+ $08	Registro interno	
+ $0A	Word di stato speciale	
+ $0C	Stadio di pipeline d'istruzione C	
+ $0E	Stadio di pipeline d'istruzione B	
+ $10	Indirizzo di difetto di ciclo di dati	
+ $12		
+ $14	Registro interno	
+ $16	Registro interno	
+ $18	Buffer di uscita dei dati	
+ $1A		
+ $1C	Registro interno	
+ $1E	Registro interno	

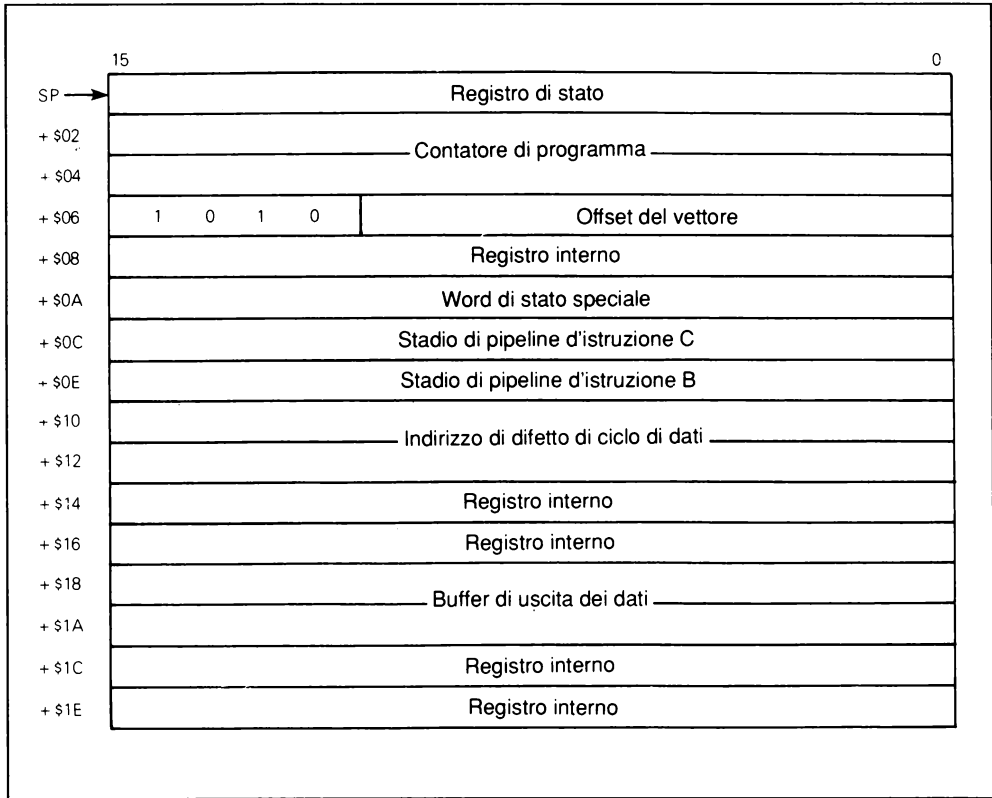



Fig. 11.12 *Frame di stack di difetto di ciclo di bus corto.*

Frame di stack di difetto di ciclo di bus lungo (46 word). La Fig. 11.13 mostra il frame di stack lungo creato quando si verifica un errore di bus nel mezzo di un'istruzione che la CPU sta eseguendo. La CPU salva il suo intero stato allo scopo di proseguire l'esecuzione, se necessario, dopo che la routine di gestione dell'errore di bus ha corretto la situazione che ha generato l'eccezione.

	15 ... 0
SP →	Registro di stato
+ $02	Contatore di programma
+ $06	1 0 1 1 — Offset del vettore
+ $08	Registro interno
+ $0A	Word di stato speciale
+ $0C	Stadio di pipeline d'istruzione C
+ $0E	Stadio di pipeline d'istruzione B
+ $10 / + $12	Indirizzo di difetto di ciclo di dati
+ $14	Registri interni, 2 word
+ $18	Buffer di uscita dei dati
+ $1C / + $22	Registri interni, 4 word
+ $24	Indirizzo dello stadio B
+ $28 / + $2A	Registri interni, 2 word
+ $2C	Buffer d'ingresso dei dati
+ $30 / + $5A	Registri interni, 22 word

Fig. 11.13 *Frame di stack di difetto di ciclo di bus lungo.*

Frame di stack miscellanei. Un frame di stack “momentaneo” viene creato nello stack d'interruzione quando lo stack principale è impiegato per l'elaborazione dell'interruzione. Esso è simile al frame di stack fondamentale di quattro word, ma ha un diverso codice di formato. Un frame di stack di 10 word viene creato per certe eccezioni relative al coprocessore.

11.7.2 Priorità di eccezione

Le eccezioni possono essere classificate in base alle loro priorità. Tali priorità sono prestabilite dal progettista del processore e non possono essere modificate. Per determinare la priorità, la Motorola suddivide le eccezioni in cinque gruppi, come mostrato nella Tab. 11.8. L'eccezione di massima priorità è quella di reset nel gruppo 0, che causa l'inizializzazione del sistema come descritto nel cap. 10. Se si verifica questa eccezione, qualsiasi altra elaborazione in corso viene terminata immediatamente. Tutte le altre eccezioni vengono elaborate in base alla rispettiva priorità, se due o più di esse si presentano contemporaneamente.

Le eccezioni del gruppo 1 hanno la precedenza su tutte le altre eccezioni tranne quella di reset. Viene sospesa perfino l'elaborazione dell'eccezione per il salvataggio nello stack delle informazioni di un'eccezione di priorità inferiore allorché viene riconosciuto un errore di indirizzo o un errore di bus. Quindi, le eccezioni del gruppo 1 saranno elaborate e gestite prima di elaborare e gestire eccezioni di priorità inferiore.

Quando l'eccezione è causata da un'istruzione del gruppo 2, l'elaborazione avviene come parte dell'esecuzione dell'istruzione. Tale esecuzione sarà sempre completata a meno che non si presenti un'eccezione di reset o del gruppo 1. Se il programma ed il sistema stanno operando correttamente, soltanto un'eccezione di reset potrebbe causare la terminazione prematura dell'elaborazione e della gestione di queste eccezioni.

Nel gruppo 3, non viene eseguita alcuna istruzione che causa un'eccezione; invece, viene intrapresa l'elaborazione dell'eccezione e l'esecuzione prosegue nella routine di gestione dell'eccezione, se nessuna eccezione di priorità superiore viene rivelata durante l'elaborazione di quella corrente.

Le eccezioni del gruppo 4 non possono presentarsi finché non sia stata completata l'esecuzione dell'istruzione per qualunque istruzione che causa un'eccezione. Tuttavia, se un'interruzione viene riconosciuta quando un'interruzione di traccia è in sospeso, avviene dapprima l'elaborazione dell'eccezione per la traccia, seguita dall'elaborazione dell'eccezione per l'interruzione. Questo è il significato delle sottopriorità per le eccezioni del gruppo 4, come mostrato nella Tab. 11.8. Comunque, quando l'elaborazione normale prosegue dopo il salvataggio nello stack delle informazioni di traccia e d'interruzione, l'esecuzione continua nel gestore d'interruzione. Al completamento della routine d'interruzione, il controllo passa alla routine di gestione della traccia.

Tab. 11.8 *Priorità di eccezione.*

Gruppo/ priorità	Eccezione e priorità relativa		Caratteristiche
0	0.0	Reset	Causa l 'aborto di tutta l'elaborazione, senza salvare il vecchio contesto.
1	1.0	Errore d'indirizzo	Sospende l'elaborazione (istruzione o eccezione) e salva il contesto interno.
	1.1	Errore di bus	
2	2.0	BKPT #N, CALLM, CHK, CHK2, cp media-istruzione, cp violazione di protocollo, cpTRAPcc, divisione per zero,RTE, RTM, TRAP #N, TRAPV	L'elaborazione dell'eccezione fa parte dell'esecuzione dell'istruzione.
3	3.0	Istruzione illegale, istruzione non implementata, violazione di privilegio, cp pre—istruzione	L'elaborazione dell'eccezione inizia prima che l'istruzione sia eseguita.
4	4.0	cp post—istruzione	L'elaborazione dell'eccezione inizia al termine dell'elaborazione dell'istruzionecorrente o dell'eccezione precedente.
	4.1	Traccia	
	4.2	Interruzione	

Nota: 0.0 è la priorità massima, 4.2 è la minima.

ESERCIZI

11.7.1

Si descriva il modo in cui la routine di gestione dell'eccezione dell'errore di bus potrebbe funzionare in un sistema con memoria virtuale, dopo che è avvenuto un difetto di pagima.

11.7.2

Si descriva l'attività che ha luogo a causa del verificarsi di ciascuna delle seguenti condizioni:

(a) Le eccezioni di trappola, di traccia e d'interruzione sono simultaneamente in sospeso.
(b) Un errore di bus si presenta durante l'elaborazione per una traccia.
(c) Un'istruzione illegale viene riconosciuta mentre è abilitata una singola istruzione di traccia.

CAPITOLO 12

APPLICAZIONI AVANZATE DELL'MC68020

In questo capitolo saranno trattate diverse caratteristiche dei computer basati sull'MC68020 che sono state concepite per migliorare le prestazioni del sistema. Gli argomenti includono la memoria cache della CPU, l'interfaccia di coprocessore, il supporto del sistema operativo e le applicazioni di multielaborazione. Innanzitutto, nel par. 12.1, sarà trattata la memoria cache sul chip. Questa memoria ad alta velocità fa aumentare la velocità di esecuzione di un programma quando le istruzioni sono contenute in un cache. La CPU ha un'interfaccia di coprocessore ed un gruppo di istruzioni per consentire a coprocessori speciali o a coprocessori della Motorola di essere aggiunti ad un sistema di computer. Dopo che le caratteristiche generali dell'interfaccia e delle istruzioni di coprocessore saranno state trattate nel par. 12.2, l'unità in virgola mobile MC68881 e l'unità di gestione della memoria impaginata (*Paged Memory Management Unit:* PPMU), entrambe prodotte dalla Motorola, saranno descritte nei capp. 12.3 e 12.4, rispettivamente.

I sistemi operativi possono impiegare le caratteristiche dell'MC68020 nella applicazioni mutliprogrammate. L'impiego dello stack principale da parte di questi sistemi operativi sarà descritto nel par. 12.5. Infine, il par. 12.6 tratta l'utilizzazione dell'MC68020 nei sistemi multiprocessore. In tutto il capitolo, saranno posti in evidenza gli aspetti di programmazione dell'interfaccia di coprocessore, i sistemi multiprogrammati e i sistemi multiprocessore. Nel cap. 13 saranno descritte varie caratteristiche di progetto circuitale e d'interfacciamento riguardanti i sistemi basati sull'MC68020 che incorporano tali capacità avanzate.

L'insieme di istruzioni in linguaggio assembler dell'unità di gestione della memoria impaginata MC68851 e del coprocessore in virgola mobile MC68881 sono presentate nell'app. C. In questa appendice sono descritte le operazioni delle istruzioni e la sintassi dell'assemblatore per entrambi i coprocessori. L'insieme di istruzioni completo dell'MC68020, comprese le istruzioni di coprocessore, sono elencate nell'app. C e definite in dettaglio nell'app. D, tratta dal manuale per l'utente del microprocessore a 32 bit MC68020 della Motorola.

12.1 LA MEMORIA CACHE DELL'MC68020

L'MC68020 contiene una memoria cache su chip di 256 byte per memorizzare le istruzioni così come vengono prelevate dalla memoria principale. Una volta che il cache è stato abilitato e riempito, la CPU preleva un'istruzione dal cache anziché dalla memoria principale durante un prelievo successivo, se tale istruzione è contenuta nella memoria cache. Ciò riduce il tempo di prelievo minimo da tre cicli di clock per un'istruzione di 32 bit a due cicli di clock. Inoltre, il bus di dati esterno del coprocessore è disponibile per i trasferimenti di dati e per altre operazioni se un'istruzione viene prelevata dal cache. Poiché l'unità del controllore di bus della CPU e la sua unità di esecuzione sono indipendenti, come descritto nel par. 4.1, il controllore di bus può trasferire i dati sul bus di sistema mentre viene eseguita l'istruzione prelevata dal cache. Dopo aver descritto l'organizzazione della memoria cache, saranno svolte alcune considerazioni di programmazione per la memoria cache.

12.1.1 Descrizione della memoria cache sul chip dell'MC68020

La memoria cache di istruzioni dell'MC68020 consiste di 64 entrate (locazioni) di longword disposte come mostrato nella Fig. 12.1[1] Quando il contatore di programma della CPU genera un indirizzo di memoria (rappresentato da A00-A31 nella figura) con cache abilitato, viene usato il campo dell'indice (A02-A07) per selezionare una delle 64 entrate nel cache. Quindi i bit d'indirizzo A08-A31 ed il bit del codice di funzione FC2 sono confrontati col "contrassegno" dell'entrata selezionata.[2] Se esiste una corrispondenza di indirizzo e modalità della CPU e il bit di validità V è posto a {1}, si dice che è avvenuto un "colpo riuscito" (*hit*) nel cache. In questo caso, l'istruzione viene passata al pipeline interno dell'MC68020 per essere eseguita.

Se non esiste corrispondenza tra il valore del contrassegno e dell'indirizzo ed il bit di codice di funzione, o se il bit di validità è {0}, allora si è in presenza di un "colpo fallito" (*miss*) e l'istruzione dev'essere prelevata dalla memoria principale tramite il bus dati della CPU. La nuova istruzione viene scritta automaticamente nell'entrata del cache ed il bit di validità è posto a {1} se l'entrata nel cache dev'essere aggiornata. Questo caso si presenta quando il cache non è "congelato", come sarà discusso nel prossimo sottoparagrafo. Per un'istruzione di una sola word, il processore preleva ancora una longword dalla memoria per aggiornare entrambe le word dell'entrata di cache di 32 bit. Se l'istruzione ha diverse word di estensione, sarebbero necessari dei prelievi multipli per registrare l'istruzione completa nella memoria cache.

[1] Soltanto le istruzioni sono memorizzate dall'MC68020 nella sua memoria cache. Sia le istruzioni che i dati possono essere memorizzati sul cache dall'MC68030, come si vedrà nel cap. 13.

[2] L'impiego dei codici di funzione per determinare la modalità di utente o di supervisore della CPU sarà descritto nel par. 13.4.

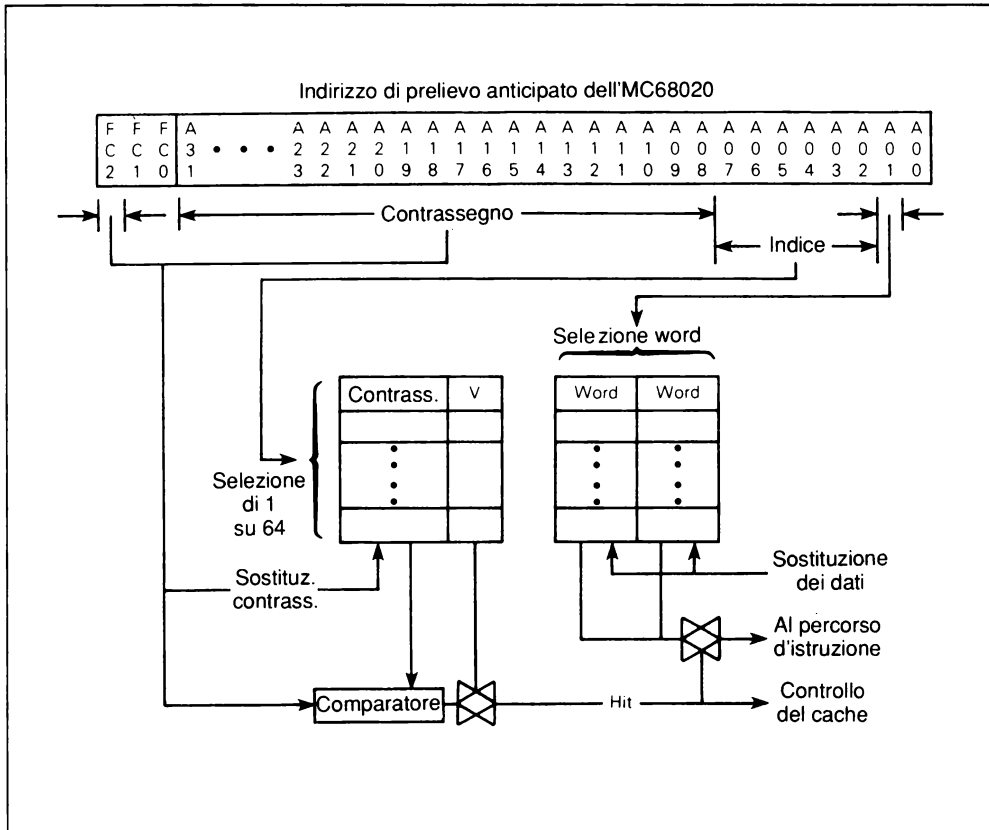



Fig. 12.1 *La memoria cache dell'MC68020. (Per gentile concessione di Motorola, Inc.)*

12.1.2 Programmazione della memoria cache dell'MC68020

La memoria cache non è direttamente accessibile dalle istruzioni di programma, ma le operazioni del cache possono essere controllate programmando il registro di controllo del cache (CACR) mostrato nella Fig. 12.2(a). Nell'MC68020, soltanto i 4 bit meno significativi sono usati. Il significato e l'uso di questi bit è indicato nella Fig. 12.2(b). Per manipolare i bit, viene utilizzata l'istruzione MOVEC (*MOVE Control register:* trasferisci registro di controllo) da un programma operante nel modo di supervisore, nella forma seguente:

```
MOVEC     <Rn>,CACR
```

dove <Rn> è un qualunque registro d'indirizzo o di dati che contiene la configurazione di bit desiderata per CACR[3:0]. Questa istruzione è stata descritta nel par. 10.2.

31		8	7							0
0		0	0	0	0	0	C	CE	F	E

C = Clear Cache
CE = Clear Entry
F = Freeze Cache
E = Enable Cache

(a)

Bit di CACR	Significato	Azione quando posto a {1}
Enable (E)	Abilitazione	Attivato dal sistema operativo dopo un reset per abilitare il cache.
Freeze cache (F)	Congelamento del cache	Impedisce l'ulteriore aggiornamento del cache.
Clear Entry (CE)	Cancellazione di entrata	Invalida una singola entrata; la CPU azzera il bit di validità per tale entrata.
Clear cache (C)	Cancellazione del cache	Invalida ogni entrata nel cache; la CPU azzera tutti i bit di validità.

(b)

31 … 8	7 … 2	1 0
Indirizzo di funzione del cache	Indice	

(c)

Fig. 12.2 *(a) Il registro di controllo del cache dell'MC68020; (b) i bit del registro di controllo del cache; (c) il registro d'indirizzo del cache dell'MC68020.*

Il bit di abilitazione E dev'essere posto a {1} per consentire al cache di operare. Di solito un sistema operativo abilita il cache durante l'inizializzazione, poiché E = {0} dopo il reset del processore. Se è necessario costringere la CPU ad accedere alla memoria esterna senza l'impiego del cache, un programma supervisore deve porre E = {0} se il cache è già abilitato. Ciò potrebbe essere desiderabile quando un segmento di programma in fase di debugging viene anche osservato da

un analizzatore di circuito che registra l'attività dei segnali del bus di sistema mentre preleva le istruzioni dalla memoria principale. Se il cache fosse abilitato, sarebbe difficile seguire la sequenza delle istruzioni a causa di quelle contenute nel cache, poiché gli accessi alla memoria principale non avverrebbero per ogni ciclo di prelievo delle istruzioni. In alternativa, il cache può essere disabilitato dalla circuiteria esterna usando una linea di segnale di abilitazione, come si discuterà nel par. 13.4.

Se il bit di "congelamento" (*Freeze:* F) vale {1} ed il cache è abilitato, un colpo mancato non causerà la sostituzione di un'entrata con un elemento prelevato dalla memoria. Invece, nel caso di un colpo riuscito, gli elementi validi saranno presentati al pipeline della CPU. Quindi un cache con F = {1} tratterrà le medesime istruzioni mentre la CPU esegue un programma. Ciò si potrebbe utilizzare per scopi di test o per costringere un segmento critico di codice a rimanere nel cache per migliorare la velocità di esecuzione complessiva di un programma.

Un programma che assegna il valore {1} al bit di "cancellazione" (*Clear:* C) del cache fa sì che tutti le entrate nel cache siano designate come non valide. Di conseguenza, la CPU porrà V = {0} per ogni entrata. La memoria cache viene solitamente cancellata da una routine del sistema operativo quando un segmento di programma o un task ha completato l'esecuzione ed il controllo sta per essere passato ad un altro task. In certi sistemi, i due task potrebbero utilizzare i medesimi indirizzi, il che potrebbe causare una certa confusione se le entrate del cache non fossero invalidate prima che il controllo sia passato al nuovo task.[3]

Il bit di cancellazione di entrata (*Clear Entry:* CE) è utilizzato da un programma insieme col registro d'indirizzo del cache (CAAR), per cancellare una singola entrata nel cache quando CE = {1}. La CPU risponde azzerando il bit di validità per l'entrata selezionata. Prima che il bit CE venga posto a {1} scrivendo nel CACR, l'indirizzo dell'entrata da cancellare dovrà essere caricato nel CAAR dall'istruzione MOVEC. Questa funzione potrebbe essere utilizzata quando il sistema operativo inserisce un'istruzione di breakpoint (BKPT) nel flusso di istruzioni. Quando viene utilizzata l'istruzione di breakpoint, ci si attende che la CPU prelevi le istruzioni dalla memoria, in modo che la circuiteria esterna possa controllare l'operazione di breakpoint come discusso in precedenza nel cap. 11.3.

ESERCIZI

12.1.1

Si descriva l'impiego della memoria cache nei seguenti casi:

(a) Il programma ha un certo numero di istruzioni di salto (*branch* o *jump*) e di ritorno.
(b) Un analizzatore di stati logici viene usato per osservare l'attività dei segnali sul bus di sistema.
(c) Le interruzioni si presentano frequentemente in un sistema.

Esiste sempre un vantaggio nell'impiego della memoria cache durante l'esecuzione del programma?

[3] Questi indirizzi sono definiti *logici* o *virtuali*. È necessaria un'unità di gestione della memoria per convertirli in indirizzi fisici nella memoria. Questo argomento sarà approfondito nel par. 12.4.

12.1.2

Si scrivano i segmenti di istruzioni per svolgere le seguenti azioni:

(a) Cancellazione della memoria cache.
(b) Cancellazione dell'entrata di cache per l'indirizzo $30726.
(c) Congelamento delle entrate nella memoria cache.

12.2 PROTOCOLLO DI COPROCESSORE

In questo paragrafo sarà descritto il protocollo necessario a programmare ed utilizzare un coprocessore speciale. Tale coprocessore viene progettato ed inserito in un sistema per svolgere certe operazioni speciali richieste da un'applicazione. La Fig. 12.3 mostra un diagramma semplificato del coprocessore connesso al bus della CPU. Lo scopo è quello d'implementare un sistema per un'applicazione specifica utilizzando la CPU ed un coprocessore appropriato progettato appositamente. Anche i coprocessori "standard" della Motorola operano nel modo qui descritto. Tuttavia, per quanto concerne l'unità di gestione della memoria impaginata dell'MC68851 e per i coprocessori in virgola mobile MC68881 e MC68882, le istruzioni eseguite appaiono come estensioni dell'insieme di istruzioni dell'MC68020.[4] Non è necessario che un programmatore conosca i dettagli dell'interfaccia di coprocessore o del protocollo di comunicazione per questi coprocessori standard. Tali dettagli saranno descritti nei parr. 12.3 e 12.4.

Una conoscenza dettagliata dell'interfaccia e del protocollo del coprocessore è necessaria per un progettista che intenda includere un coprocessore speciale in un sistema basato sull'MC68020. Inoltre, tali dettagli di funzionamento sono importanti se un coprocessore standard della Motorola è impiegato in un sistema con una CPU diversa dall'MC68020 o dall'MC68030. Disponendo di un coprocessore speciale, il programmatore può comunicare con esso mediante istruzioni incluse nell'insieme dell'MC68020. Queste istruzioni di coprocessore costituiscono l'argomento del presente paragrafo. Se un'altra CPU viene impiegata con un coprocessore MC68851 o MC68881, il programmatore deve emulare l'insieme di istruzioni del coprocessore standard della Motorola quando un coprocessore della Motorola è incluso nel sistema.

Questo paragrafo descrive dapprima il modello di programmazione per un coprocessore speciale, dopodiché saranno discusse le eccezioni generate dal coprocessore. In sistemi che dispongono di un coprocessore speciale, la CPU svolge vari servizi, quali il prelievo di operandi e la gestione di eccezioni in conformità con i requisiti del processore, mentre quest'ultimo esegue le proprie istruzioni. I risultati dell'esecuzione delle istruzioni del coprocessore sono definiti interamente dal progetto del coprocessore speciale.

[4] Gli assemblatori della Motorola e di altre società per l'MC68020 assemblano le istruzioni per i coprocessori standard. Il linguaggio-macchina per i coprocessori speciali dev'essere creato dal programmatore seguendo le convenzioni della Motorola per tali istruzioni.

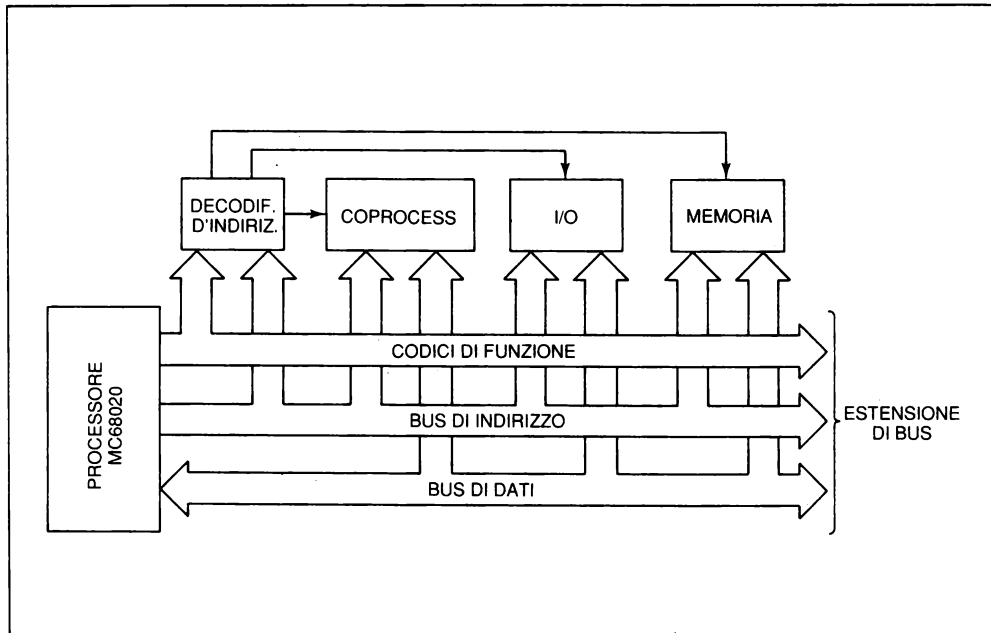



Fig. 12.3 *Connessione del coprocessore al bus della CPU.*

12.2.1 Funzionamento del coprocessore e considerazioni di programmazione

La CPU ed il coprocessore comunicano mediante i registri che fanno parte del coprocessore. Per iniziare un'operazione del coprocessore, dev'essere eseguita un'istruzione di coprocessore. Questo riceve l'istruzione dalla CPU, che indirizza il registro di comando del coprocessore nello spazio di CPU riservato ad esso, usando il formato d'indirizzo illustrato nella Fig. 12.4(a). Gli indirizzi possibili sono mostrati nella Fig. 12.4(b) per gli otto coprocessori che possono essere collegati ad un sistema basato sull'MC68020. L'identificatore di coprocessore (CP-ID) 000 è riservato all'MC68851, mentre un CP-ID di 001 è riservato l'unità in virgola mobile della Motorola. Gli altri codici d'identificazione (002-007) possono essere usati per coprocessori speciali. Dopo che il coprocessore ha accettato il comando della CPU, esso risponde ponendo una configurazione di bit, nota come *codice di risposta*, nel proprio registro di rispsta. Dopodiché, la CPU legge tale risposta ed esegue il servizio appropriato.

La sequenza generale di comunicazione tra il coprocessore e la CPU è mostrata nella Fig. 12.5. La sequenza inizia allorché un programma esegue un'istruzione di linea F. Quando un'istruzione di linea F viene riconosciuta dalla CPU, essa non viene decodificata dalla CPU ma viene passata al registro di comando del coprocessore. Le comunicazioni tra la CPU ed il coprocessore proseguono usando

Codice di funzione

Bus d'indirizzo

31	19	15	12	4 0
1 1 1	0 0 0 0 0 0 0 0 0 0 0 0	0 0 1 0	Cp-ID	0 0 0 0 0 0 0 0	Registro CIR

Campo del tipo di spazio della CPU

(a)

Spazio d'indirizzi della CPU

Indirizzo	Contenuto	
20000	Insieme reg. d'interfaccia	Spazio d'indirizzi per coprocessore con Cp-ID = 0
2001F	Riservati	
22000	Insieme reg. d'interfaccia	Spazio d'indirizzi per coprocessore con Cp-ID = 1
2201F	Riservati	
24000		
2E000	Insieme reg. d'interfaccia	Spazio d'indirizzi per coprocessore con Cp-ID = 7
2E01F	Riservati	

(b)

Note:

1. Lo spazio della CPU è definito dalle linee di segnale dei codici di funzione della CPU, come descritto nel par. 13.4.
2. L'identificazione del coprocessore (Cp-ID) è codificata nei bit 9-11 dell'istruzione di linea F che indirizza il coprocessore.

Fig. 12.4 *Lo spazio d'indirizzamento della CPU MC68020. (b) Gli indirizzi del coprocessore nello spazio della CPU.*

l'apposito registro d'interfaccia del coprocessore (*Coprocessor Interface Register:* CIR). Durante l'esecuzione di un'istruzione, il coprocessore può richiedere vari servizi dalla CPU, ponendo il codice appropriato nel CIR di risposta. Questi codici, denominati *primitive di risposta del coprocessore*, rappresentano lo stato o una richiesta proveniente dal coprocessore.

Dopo aver completato la propria istruzione, il coprocessore segnala il fatto nel registro di risposta, per cui la CPU potrà continuare l'elaborazione eseguendo l'istruzione successiva del programma.[5]

[5] L'elaborazione concorrente della CPU e del coprocessore è possibile, come descritto nel *MC68020 User's Manual.* Nella sequenza mostrata in Fig. 12.5, la CPU non può continuare ad eseguire le istruzioni finché il coprocessore non ha terminato.

Processore principale		Coprocessore
M1 Riconosce la word di operazione di linea F dell'istruzione del coprocessore		
M2 Scrive la word di comando del coprocessore nel CIR di comando	→	C1 Decodifica la word di comando ed avvia l'esecuzione del comando
		C2 Mentre è richiesto il servizio del coprocessore, esegue i passi 1 e 2 elencati di seguito:
M3 Legge il codice della primitiva di risposta del coprocessore dal CIR di risposta	↔	1) Richiede il servizio ponendo il codice appropriato di primitiva di risposta nel CIR di risposta
1) Esegue il servizio richiesto dalla primitiva di risposta 2) Se la primitiva di risposta del coprocessore indica "Ancora", vai a M3 (v. Nota 1)		2) Riceve il servizio dal processore principale
		C3 Rispecchia "Non ancora" nel CIR di risposta
		C4 Completa l'esecuzione del comando
		C5 Rispecchia lo stato di "Elaborazione terminata" nel CIR di risposta
M4 Procede con l'esecuzione dell'istruzione successiva (v. Nota 2)		

Nota: "Ancora" indica che il coprocessore sta richiedendo ancora il servizio del processore principale.

Fig. 12.5 *Comunicazione tra la CPU ed il coprocessore durante l'esecuzione di un'istruzione generale del coprocessore.*

L'insieme di registri del coprocessore. La struttura d'indirizzi richiesta di un insieme di registri del coprocessore è mostrata nella Fig. 12.6. Questi registri sono indirizzati dal loro offset dall'indirizzo di base nello spazio della CPU. L'indirizzo di base è definito dalla CPU sul proprio bus di indirizzi, nel campo di spazio della CPU (bit 16-19) della Fig. 12.4(a). Quindi, il particolare registro d'interfaccia del coprocessore è selezionato dai bit 0-4. La CPU seleziona automaticamente il registro appropriato quando viene eseguita una delle istruzioni del coprocessore.[6]

[6] Il protocollo per un'istruzione di coprocessore è determinato da una sequenza microcodificata entro la CPU. Le operazioni effettive eseguite e la quantità di dati trasferiti dipendono dalla scopo del coprocessore.

	31	15 0
00	Risposta*	Controllo*
04	Salvataggio*	Ripristino*
08	Word di operazione	Comando*
0C	(Riservato)	Condizione*
10	Operando*	
14	Selezione di registro	(Riservato)
18	Indirizzo di istruzione	
1C	Indirizzo di operando	

Nota: I registri contrassegnati da un asterisco (*) sono richiesti per qualunque coprocessore.

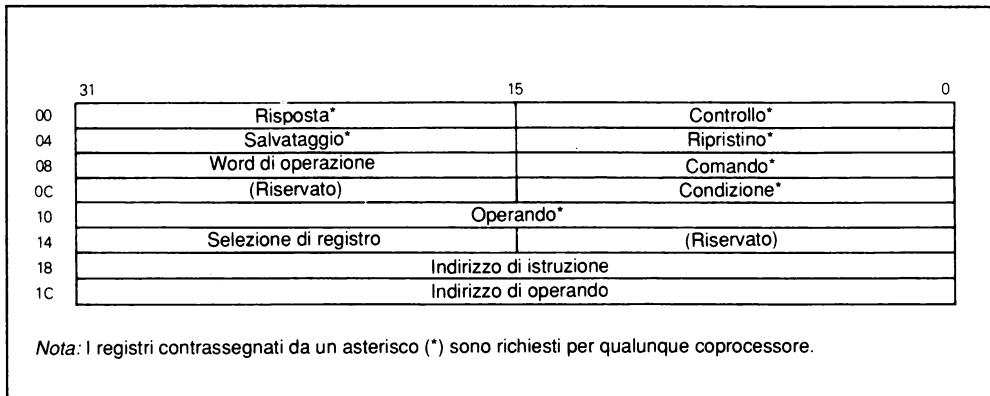



Fig. 12.6 *L'insieme di registri di un coprocessore speciale.*

Non è necessario che tutti i registri siano presenti per un coprocessore specifico, ma sono richiesti quelli contrassegnati con un asterisco nella Fig. 12.6.

Le istruzioni di coprocessore dell'MC68020. Le istruzioni per un coprocessore speciale sono classificabili in quattro categorie: generali, condizionali, di salvataggio del contesto, e di ripristino del contesto. Queste istruzioni si distinguono per il tipo di operazione svolta dal coprocessore, ma non determinano il risultato specifico, poiché questo dipende dal progetto del coprocessore. Un'istruzione in una categoria particolare determina i registri d'interfaccia del coprocessore a cui può accedere la CPU e definisce il protocollo di comunicazione tra la CPU ed il coprocessore.

La Tab. 12.1 elenca le istruzioni specifiche del coprocessore che fanno parte dell'insieme di istruzioni dell'MC68020. Nella tabella è elencato anche l'impiego tipico di queste istruzioni. L'istruzione generale cpGEN ha il formato mostrato nella Fig. 12.7. La word di operazione dev'essere codificata come mostrato, in cui l'indirizzo effettivo definisce le locazioni di qualsiasi operando al di fuori del coprocessore. Il modo d'indirizzamento specifico è una delle modalità dell'MC68020 definite nel cap. 5. Comunque, la modalità è determinata interamente dai requisiti del coprocessore. Le comunicazioni tra la CPU ed il coprocessore per un'istruzione generale sono state illustrate nella Fig. 12.5. Il formato delle altre istruzioni di coprocessore è definito nell'app. D.

Risposta del coprocessore. Le primitive di risposta del coprocessore sono codificate in una word di 16 bit che la CPU legge dal registro di risposta del coprocessore. Come mostrato nella Fig. 12.8, la primitiva di risposta è suddivisa in due campi che determinano un'azione specifica, e 3 bit (13-15) che controllano altre operazioni. Il coprocessore deve presentare una delle primitive di risposta consentite per ciascun comando del coprocessore inviato dalla CPU. Inoltre, il bit di "chiamata ulteriore" (*Come or Call Again:* CA) può essere posto a {1} dal coprocessore

Tab. 12.1 *Istruzioni del coprocessore.*

Categoria di istruzioni del coprocessore	Impiego tipico
Generale cpGEN	Elaborare i dati o svolgere altre operazioni definite dal coprocessore.
Condizionale cpBcc cpDBcc cpScc cpTRAPcc	Consentire il controllo del programma in base alle istruzioni del coprocessore ed alla risposta del coprocessore.
Salvataggio cpSAVE	Salvare lo stato del coprocessore.
Ripristino cpRESTORE	Ripristinare lo stato del coprocessore.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Cp ID			0	0	0	Indirizzo effettivo					
Comando di coprocessore															
Indirizzo effettivo opzionale o word di estensione definita dal coprocessore															

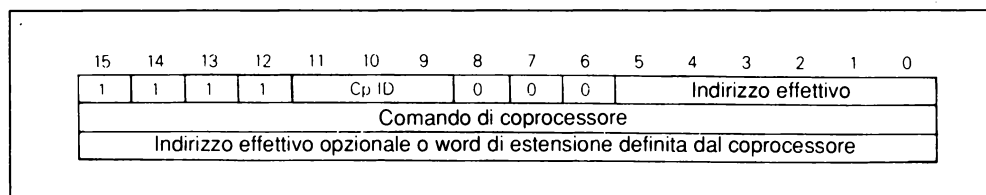



Fig. 12.7 *Formato dell'istruzione generale di coprocessore.*

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	DR	Funzione					Parametro							

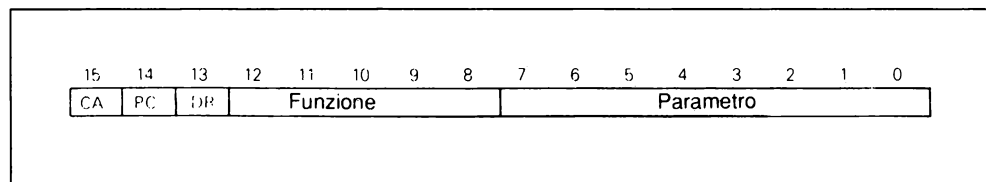



Fig. 12.8 *Primitiva di risposta del coprocessore.*

per far sì che la CPU legga più volte il registro di risposta durante l'esecuzione di una singola istruzione del coprocessore. Se il bit del contatore di programma (*Program Counter:* PC) è {1}, allora la CPU passerà il contenuto del PC al coprocessore. Il bit di direzione (*Direction:* DR) determina la direzione del trasferimento dell'operando tra la CPU ed il coprocessore.

Tab. 12.2 *Esempi di risposte del coprocessore.*

Risposta	Uso o condizione
STATO	
Occupato	La CPU dovrebbe continuare a inviare un comando di coprocessore finché questo non diviene libero.
Nullo	Lo stato del coprocessore viene presentato alla CPU.
RICHIESTA	
Valutazione e trasferimento dell'indirizzo effettivo o dell'operando al coprocessore.	Il coprocessore sta richiedendo un indirizzo o un operando alla CPU.
Trasferimento dell'operando del coprocessore alla memoria	Il coprocessore ha il valore dell'operando che la CPU dovrà registrare nella memoria.
Trasferimento del valore del registro tra la CPU ed il coprocessore	Scambio degli operandi o degli indirizzi tra l'insieme di registri della CPU ed il coprocessore.
Generazione di eccezione	Richiesta alla CPU di elaborare l'eccezione.

Un esempio delle possibili risposte del coprocessore è elencato nella Tab. 12.2.[7] La risposta di stato o una richiesta fa sì che la CPU intraprenda l'azione appropriata. Per un coprocessore speciale, la sua circuiteria interna deve presentare la risposta appropriata, per far sì che la CPU fornisca un servizio, come un trasferimento di dati, quando tale servizio viene richiesto. Per esempio, in un'operazione di trasferimento di dati al coprocessore, la CPU risponde alla richiesta scrivendo l'operando nel registro di operando del coprocessore, mostrato nella Fig. 12.6. Quando le operazioni di trasferimento di dati ed altre richieste sono servite dalla CPU, la circuiteria d'interfaccia del coprocessore dev'essere progettata in modo tale da rispondere correttamente alle linee di segnali di bus della CPU.

[7] L'insieme completo delle primitive di risposta è definito nell'*MC68020 User's Manual*.

Esempio 12-1

La Fig. 12.9 illustra la sequenza di attività quando viene eseguita l'istruzione dell'MC68881

FMOVE.P (A0)+,FP0

Questa istruzione trasferisce un valore decimale impaccato (P) lungo 12 byte dalla memoria al registro di coprocessore in virgola mobile FP0. Questo registro fa parte dell'insieme di registri programmabili dell'MC68881, che sarà descritto nel par. 12.3. Esso non è uno dei registri d'interfaccia del coprocessore. L'istruzione viene assemblata come mostrato nella Fig. 12.9(a) in un'istruzione in linguaggio-macchina di 32 bit, col formato:

$F218
$4C00

in cui la word di operazione seleziona l'MC68881 usando un'istruzione di tipo cpGEN. I 6 bit meno significativi definiscono la modalità d'indirizzamento con postincremento per il registro A0 dell'MC68020. La seconda word, $4C00, è la word di estensione richiesta per l'MC68881 per indicare il formato della sorgente di dati, il registro di destinazione FP0 nel coprocessore, e l'istruzione FMOVE stessa.

COPROCESSOR INTERFACE EXAMPLE

INSTRUCTION
(FC's)ADDRESS BUS
DATA BUS
COPROCESSOR RESPONSE

FMOVE.P (A0)+,FP0
$F218
$4C00

1 1 1 1 | 0 0 1 | 0 0 0 | 0 1 1 | 0 0 0
Line "F"
CpID (Default=001 for MC68881)
CP Instr. Type (000=General)
Mode
Register
Operand Effective Address

0 1 0 | 0 1 1 | 0 0 0 | 0 0 0 0 0 0 0
OP Class
Source Data Format
Destination Register
"FMOVE" Instruction
010=F<op>.<size> <EA>,FPn
011=.P

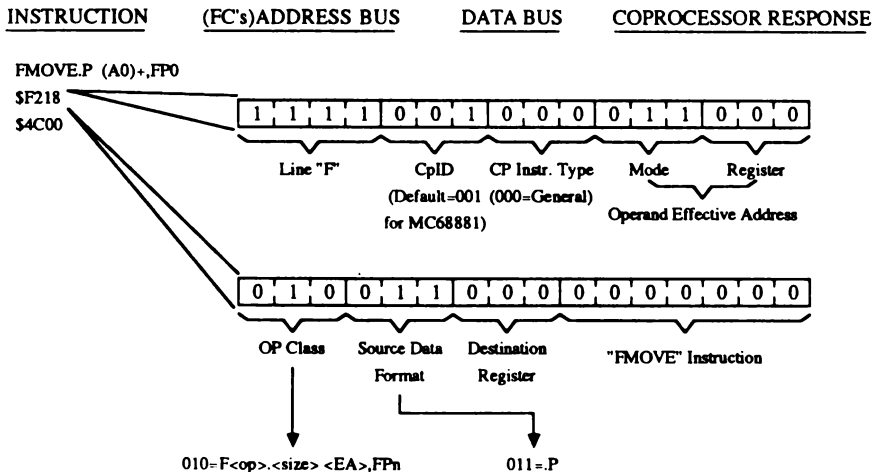



Fig. 12.9 *Esempio di istruzione di coprocessore: (a)* FMOVE.P (A0)+, FP0.

COPROCESSOR INTERFACE EXAMPLE

INSTRUCTION	(FC's)ADDRESS BUS	R/W	DATA BUS	COPROCESSOR RESPONSE
FMOVE.P (A0)+,FP0				
$F218				
$4C00				
	(111)$2200A	W	$4C00	DSACK (accept the command)
	(111)$22000	R	$960C ◄	Place Response Register on Data Bus and DSACK

1	0	0	1	0	1	1	0	0	0	0	0	1	1	0	0

Call Again

Don't Pass PC

Valid <EA> mode for Source operand:

110 = Memory

Length of Data Transferr (in bytes):

.P = 12 bytes

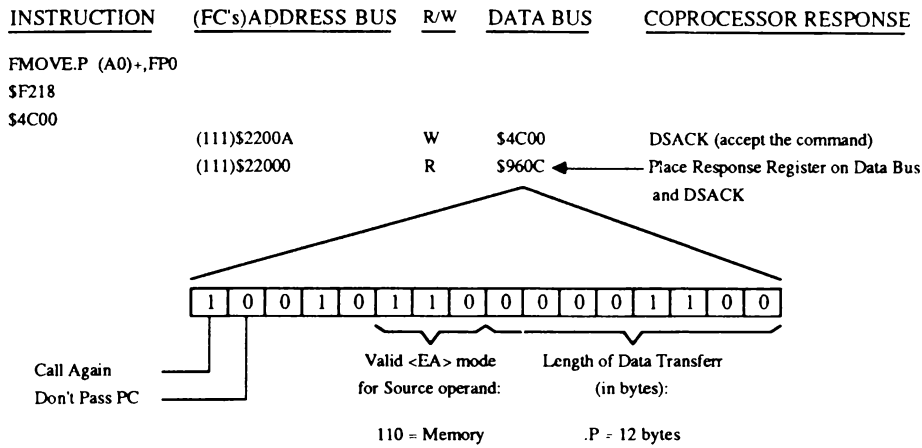



Fig. 12.9 *Esempio di istruzione di coprocessore: (b) Risposta del coprocessore all'istruzione* FMOVE.

COPROCESSOR INTERFACE EXAMPLE

INSTRUCTION	(FC's)ADDRESS BUS	R/W	DATA BUS	COPROCESSOR RESPONSE
FMOVE.P (A0)+,FP0				
$F218				
$4C00				
	(111)$2200A	W	$4C00	DSACK (accept the command)
	(111)$22000	R	$960C	Place Response Register on Data Bus and DSACK
	(x01)$xxxxxxxx	R	1 st 4 data bytes from memory	
	(111)$22010	W	1 st 4 data bytes	DSACK
	(x01)$xxxxxxxx	R	memory data	
	(111)$22010	W	Cp data	DSACK
	(x01)$xxxxxxxx	R	memory data	
	(111)$22010	W	Cp data	DSACK
	(111)$22000	R	$0802	Place Response Register on Data Bus and DSACK

0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0

Don't Call Again (done)

Don't Pass PC

NULL Response

True/False (ignored)

Processing finished this instruction

"Null_release" primitive

Fig. 12.9 *Esempio di istruzione di coprocessore: (c) Trasferimento completo.*

Dopo che l'MC68881 ha ricevuto l'istruzione ($4C00) nel suo registro di comando, il coprocessore risponde richiedendo alla CPU di trasferire l'operando richiesto dalla memoria. La primitiva di risposta è illustrata nella Fig. 12.9(b). Poiché il bit CA (*Call Again:* chiamata ulteriore) è posto a {1}, la CPU leggerà di nuovo il registro di risposta dopo che il trasferimento di dati sarà stato completato.[8]

Il trasferimento completo è illustrato nella Fig. 12.9(c). Quando il coprocessore riceve il valore di dati, esso risponde con una primitiva di "rilascio" nulla. A questo punto, la CPU è libera di avviare l'esecuzione dell'istruzione successiva nel programma.

12.2.2 Eccezioni di coprocessore

Un coprocessore può avere tre primitive di risposta che producono l'elaborazione di un'eccezione da parte della CPU. Queste primitive sono inviate quando il coprocessore rivela una condizione che non può risolvere. In particolare, il coprocessore può rivelare violazioni di protocollo, comandi illegali, o errori specifici di elaborazione dei dati. Le cause degli errori e la gestione delle eccezioni richiesta dalla CPU dipendono dal progetto del coprocessore. Per l'MC68851 e l'MC68881, sono riservati vettori di eccezione specifici per le eccezioni di coprocessore, come è stato spiegato nel par. 11.5.

ESERCIZI

12.2.1 Usando l'esempio 12.1, si definisca ciò che segue durante l'esecuzione dell'istruzione FMOVE.

(a) Il valore del bus di indirizzi quando quando la CPU passa le istruzioni al coprocessore.
(b) Il valore sul bus di indirizzi quando la CPU legge il registro di risposta.
(c) Il valore sul bus di indirizzi quando dev'essere trasferito l'operando.

12.2.2 Si descrivano i passi richiesti per progettare un insieme di istruzioni speciali di un coprocessore usando l'insieme di istruzioni di coprocessore dell'MC68020. Tale insieme di istruzioni è definito nell'app. D.

12.2.3 Si descrivano le differenze tra un coprocessore ed un dispositivo perfierico standard in termini sia dello spazio di indirizzi che dei requisiti di programmazione.

---

[8] Le linee di segnale interessate (R/W, DSACK) saranno spiegate nel par. 13.4. Nell'esempio, una rispo-

12.2.4

Si consideri un coprocessore progettato per eseguire la moltiplicazione vettoriale di due array nella memoria come Z(I) = X(I)*Y(I), con I = 1, ... , N. Il coprocessore dovrebbe consentire ai moltiplicatori ed ai moltiplicandi di formare prodotti di 64 bit. Per un siffatto coprocessore:

(a) Si progetti l'insieme di istruzioni come istruzioni di macroassembler, usando i nomi mnemonici appropriati.
(b) Si definisca la comunicazione tra la CPU ed il coprocessore in una forma simile a quella della Fig. 12.5.

12.3 I COPROCESSORI IN VIRGOLA MOBILE DELLA MOTOROLA

L'MC68881 e l'MC68882 sono coprocessori che svolgono operazioni matematiche in un sistema di computer. Ognuno dei due coprocessori esegue l'aritmetica in virgola mobile, che comprende l'addizione, la sottrazione, la moltiplicazione e la divisione. I coprocessori eseguono anche conversioni numeriche e calcolano il valore di varie funzioni matematiche, come il seno e il coseno. In questo paragrafo si definisce il modello di programmazione e vengono presentate alcune applicazioni del coprocessore MC68881. L'MC68882 ha un identico modello di programmazione. Questi coprocessori della Motorola impiegano il formato di virgola mobile dello standard IEEE per i numeri in virgola mobile, come già descritto nel par. 3.3. Si presume che il lettore sia già a conoscenza del materiale presentato nel cap. 3.

12.3.1 La programmazione del coprocessore MC68881

Per il programmatore in linguaggio assembler, il coprocessore in virgola mobile è rappresentato da un insieme di registri che vengono programmati mediante un insieme di istruzioni in virgola mobile predefinito. Il coprocessore stesso è incorporato in un sistema basato sull'MC68020, come descritto nel par. 12.2. Si presuppone che il lettore disponga di un assemblatore che converte le istruzioni in virgola mobile. Sia l'assemblatore della Motorola che l'assemblatore della Quelo, descritti in questo libro, riconoscono e convertono tutte le istruzioni per l'MC68881. Un programmatore in linguaggio assembler deve avere familiarità coi formati di dati dell'MC68881, con l'insieme di registri e con l'insieme di istruzioni dell'MC68881 per utilizzare con successo il coprocessore in un'applicazione.

I formati di dati del coprocessore MC68881. Come mostra la Fig. 12.10, l'MC68881 ammette i seguenti tipi di dati;

(a) Intero di byte (B)
(b) Intero di word (W)
(c) Intero di longword (L)
(d) Reale in singola precisione (S)
(e) Reale in doppia precisione (D)
(f) Reale in precisione estesa (X)
(g) Reale di stringa decimale impaccato (P)

dove la lettera tra parentesi che segue il tipo di dati rappresenta il suffisso da aggiungere alla forma del linguaggio assembler di un'istruzione. I valori di byte, di word e di longword, designati da B, W e L, rispettivamente, sono i tipi interi di dati standard dell'MC68020. I reali in singola precisione (S) e in doppia precisione (D) sono valori in virgola mobile di 32 bit e di 64 bit, rispettivamente, come definito nel par. 3.3. I valori reali in precisione estesa (X) hanno 80 bit, con un esponente di 15 bit ed una mantissa di 64 bit. Questi valori estesi sono usati internamente dall'MC68881 per mantenere la precisione delle operazioni matematiche. Se un valore esteso viene posto nella memoria da un programma, esso deve avere il formato di 96 bit mostrato nella Fig. 12.10. Nella memorizzazione dei valori in singola precisione e in doppia precisione, non viene registrato l'1 di testa nella memoria, come spiegato nel par. 3.3. Un valore di polarizzazione viene aggiunto all'esponente quando il valore in virgola mobile è rappresentato nella memoria. Il suddetto valore è 127 per la singola precisione, 1023 per la doppia precisione, e 16383 per i numeri in precisione estesa.

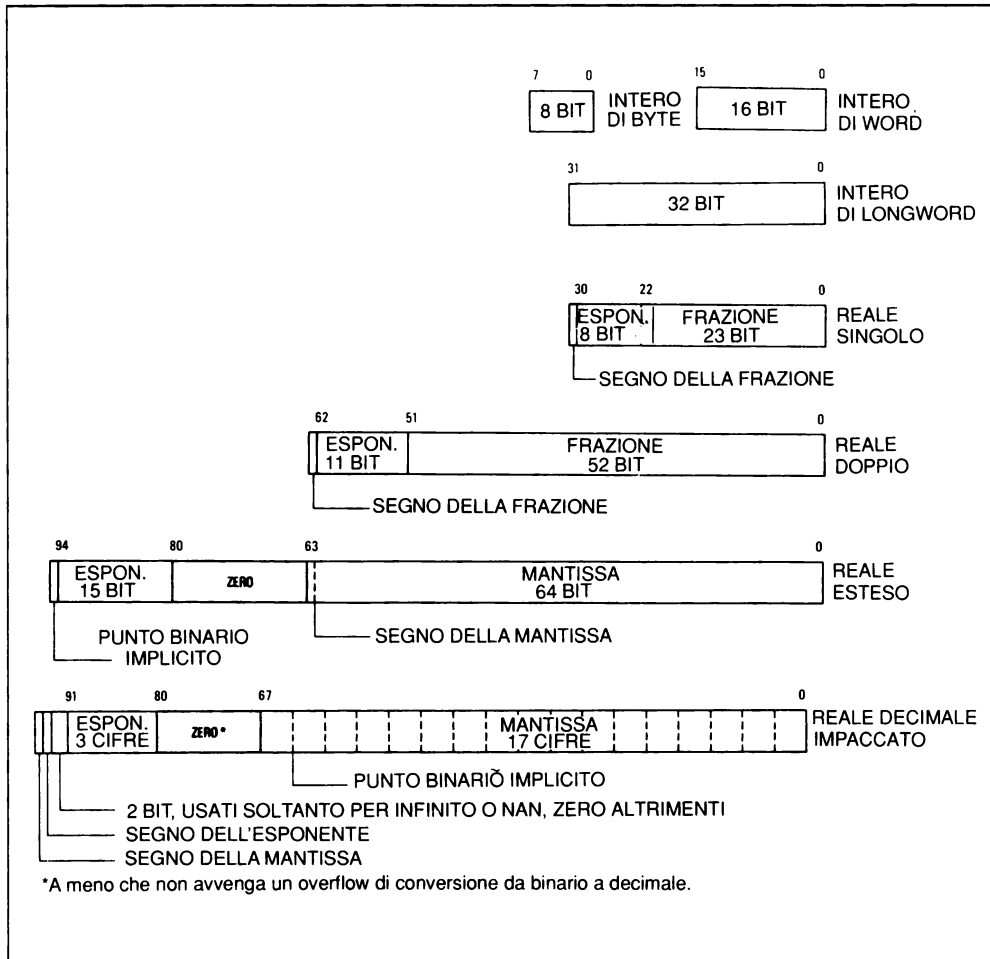



Fig. 12.10 *Formati di dati per il coprocessore in virgola mobile MC68881.*

Il valore decimale impaccato dell'MC68881 rappresenta un esponente in base 10 di tre cifre ed una mantissa in base 10 di 17 cifre, dove tutte le cifre sono nel formato BCD. Nella memoria, l'intera stringa è lunga 96 bit, sebbene i bit 68-80 siano memorizzati come zeri. Internamente l'MC68881 esegue tutte le operazioni con valori in precisione estesa, indipendentemente dal formato di dati specificato nell'istruzione del linguaggio assembler.

Esempio 12-2

Le istruzioni di programma della Fig. 12.11 impiegano un certo numero di formati di dati per l'operando di sorgente nelle istruzioni di addizione in virgola mobile (*Floating Point ADDition:* FPADD). Anche se gli assemblatori differiscono nella loro capacità di gestire l'aritmetica in virgola mobile, la direttiva FEQU (*Floating point EQUate:* uguaaglia in virgola mobile) è solitamente impiegata per definire un valore costante tramite un simbolo, come avviene per il valore GAS nell'esempio. Un valore in virgola mobile o decimale può essere specificato come un numero che contiene un punto decimale. L'esponente può essere specificato come una potenza di 10 usando il simbolo E dopo il numero, seguito dal segno e dal valore dell'esponente.

```
                             1.            TTL     FIGURA 12.11
                             2.            LLEN    100
  # 00000063                 3.  .RETURN   EQU     $0063
                             4.  *
                             5.  *   ISTRUZIONI MISCELLANEE DELL'MC68881
                             6.  *
                             7.            OPT     P=68881
00010000                     8.            ORG     $10000
                             9.  *
  d 0000230623000000        10.  GAS       FEQU.D  6.23E+23
    000000000000
00010000  F23C 5822 0000    11.            FADD.B  #0,FP0            ;BYTE
00010006  F23C 5A02 0005    12.            FADD.W  #5,FP1            ;WORD
0001000C  F23C 4122         13.            FADD.L  #100000,FP2       ;LONGWORD
          000186A0
                            14.  *
00010014  F23C 45A2 4049    15.            FADD.S  #3.14159,FP3      ;SINGOLA PRECISIONE
          0FD0
0001001C  F23C 5622 44E0    16.            FADD.D  #GAS,FP4          ;DOPPIA PRECISIONE
          7D9C B9BC 9119
00010028  F23C 4AA2 3F9B    17.            FADD.X  #1234.5E-33,FP5 ;PRECISIONE ESTESA
          0000 C84F 1FAF
          B610 6F23
00010038  F23C 4F22 0025    18.            FADD.P  #1.23E25,FP6      ;DECIMALE
          0001 2300 0000
          0000 0000
                            19.  *
00010048  4E4F              20.            TRAP    #15               ;RITORNA AL MONITOR
0001004A  0063              21.            DC.W    .RETURN
                            22.  *
0001004C                    23.            END
```

Fig. 12.11 *Esempi di formati di dati dell'MC68881.*

L'insieme di registri del coprocessore MC68881. L'insieme di registri del programmatore per l'MC68020/MC68881 consiste dei registri di dati e di indirizzi dell'MC68020 più l'insieme di registri dell'MC68881, come mostrato nella Fig. 12.12. In un'istruzione del linguaggio assembler, gli otto registri in virgola mobile sono designati come FP0, FP1, ..., FP7. Questi registri contengono i valori in precisione estesa che possono essere convertiti in altri formati quando i numeri vengono trasferiti dall'MC68881. Se un numero in un altro formato viene trasferito all'MC68881, il valore viene convertito in precisione estesa dal coprocessore prima della memorizzazione in uno dei suoi registri in virgola mobile.

I registri di controllo, di stato e di indirizzo dell'istruzione sono usati per controllare o sorvegliare l'attività dell'MC68881. Un byte nel registro di controllo è usato per abilitare o disabilitare l'elaborazione delle eccezioni quando si presentano certi errori. Il byte di controllo della modalità consente al programmatore di selezionare il metodo di arrotondamento dei risultati numerici. Un registro di stato di quattro byte contiene i bit per indicare il risultato di un trasferimento di dati o di un'operazione matematica. Vari bit indicano condizioni di errore quali overflow, underflow o divisione per zero quando viene tentata un'elaborazione aritmetica. Il registro d'indirizzo dell'istruzione contiene l'indirizzo dell'istruzione in corso di esecuzione. Esso è usato da una routine di gestione dell'eccezione in virgola mobile per determinare l'indirizzo di un'istruzione che ha causato un'eccezione. Ulteriori dettagli su questi registri speciali sono forniti nel *MC68881/MC68882 Floating-Point Coprocessor User's Manual* disponibile dalla Motorola, Inc. o dalla Prentice Hall, Inc.

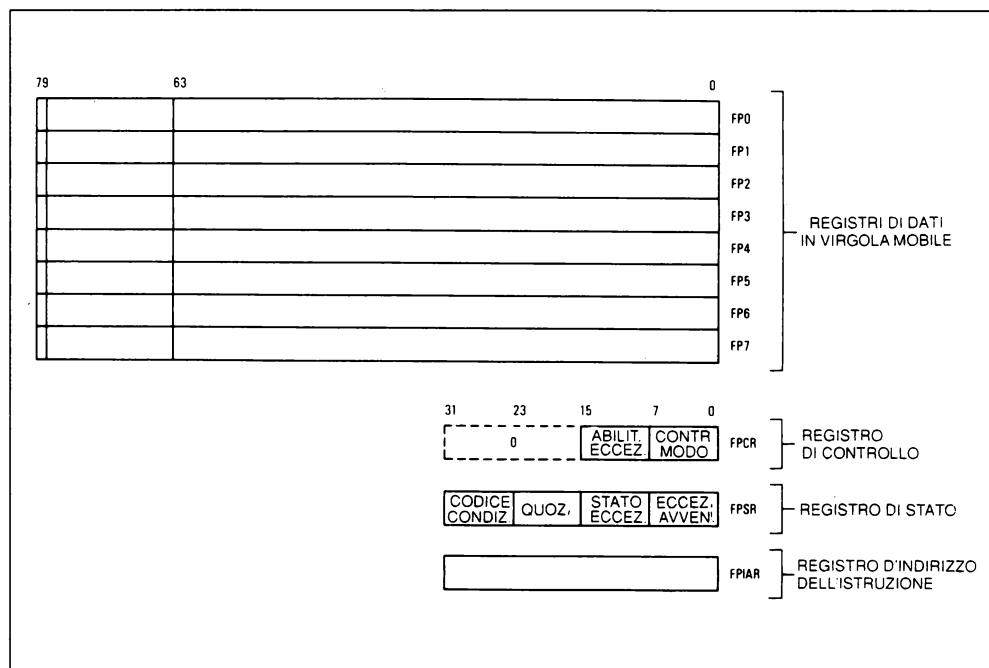



Fig. 12.12 *L'insieme di registri del coprocessore in virgola mobile MC68881.*

L'insieme di istruzioni del coprocessore MC68881. L'insieme di istruzioni del coprocessore MC68881 si può suddividere in cinque categorie:

(a) Trasferimento di dati

(b) Operazioni diadiche

(c) Operazioni monadiche

(d) Controllo del programma

(e) Controllo del sistema

Queste istruzioni consentono ad un programmatore di creare programmi sofisticati per applicazioni matematiche, tecniche o scientifiche. Le prime quattro categorie di istruzioni del coprocessore MC68881 sono presentate qui. Comunque il lettore può consultare l'*MC68881/MC68882 User's Manual* per ulteriori dettagli in merito ad una istruzione specifica. Il formato del linguaggio assembler per le istruzioni dell'MC68881 è presentato nell'app. C.

La Tab. 12.3 definisce la notazione impiegata nelle tabelle di istruzioni che saranno presentate. I suffissi dimensionali (B, W, L, S, D, X, P) in un'istruzione indicano il formato dei dati. I registri di virgola mobile FP0, FP1, ..., FP7 servono come registri di sorgente o di destinazione per operandi entro l'MC68881. La notazione generale nelle tabelle di istruzioni è FPm per un registro di sorgente e FPn per un registro di destinazione. La designazione "ccc" rappresenta una costante contenuta nella memoria ROM dell'MC68881. Tra i valori delle costanti ci sono π, *e*, e le potenze di 10.

Tab. 12.3 *Notazione dell'insieme di istruzioni (MC68881/MC68882)*

B, W, L	La stessa dimensione della famiglia di processori dell'MC68000; specifica un tipo di dati intero con segno (in complemento a 2) di byte (8 bit), word (16 bit) o longword (32 bit).
S	Formato di dati in virgola mobile in singola precisione (32 bit).
D	Formato di dati in virgola mobile in doppia precisione (64 bit).
X	Formato di dati in virgola mobile in precisione estesa (96 bit, 16 bit inutilizzati).
P	Formato di dati in virgola mobile BCD impaccato (96 bit, 12 byte).
FPm, FPn	Uno degli otto registri di dati in virgola mobile.
FPcr	Uno dei tre registri di controllo di sistema in virg. mob. (FPCR, FPSR o FPIAR).
<EA>	Qualsiasi valida modalità d'indirizzamento dell'MC68020.
K	Un intero con segno in complemento a 2 (da –64 a +17) che specifica il formato di un numero da memorizzare in forma decimale BCD impaccata.
ccc	Un indice nella ROM di costanti dell'MC68881.
<lista>	Una lista di registri di dati o di controllo in virgola mobile.
<etichetta>	Un'etichetta relativa usata da un assemblatore per calcolare uno spostamento.

Tab. 12.4 *Istruzioni di trasferimento dei dati (MC68881/MC68882).*

Istruzione	Sintassi dell'operando	Formato dell'operando	Operazione
FMOVE	FPm,FPn	X	Sorgente → destinazione
	<EA>,FPn	B, W, L, S, D, X, P	
	FPm,<EA>	B, W, L, S, D, X	
	FPm,<EA>{#K}	P	
	FPm,<EA>{Dn}	P	
	<EA>,FPcr	L	
	FPcr,<EA>	L	
FMOVECR	#ccc,FPn	X	Costante di ROM → FPn
FMOVEM	<EA>,<lista>[1]	L, X	Registri elencati → destinazione
	<EA>,Dn	X	
	<lista>[1],<EA>	L,X	Sorgente → registri elencati
	Dn,<EA>	X	

Nota: La lista di registri può includere qualunque combinazione degli otto registri in virgola mobile, o può contenere qualunque combinazione dei tre registri di controllo FPCR, FPSR, e FPIAR. Se la maschera della lista di registri risiede in un registro di dati del processore principale, soltanto i registri di dati in virgola mobile possono essere specificati.

Le istruzioni di trasferimento di dati per il coprocessore MC68881 sono elencate nella Tab. 12.4. In tale lista, FMOVE e FMOVEM (trasferimento multiplo) sono simili alle istruzioni MOVE e MOVEM, rispettivamente, dell'MC68020. L'operando di sorgente dell'istruzione FMOVE può essere contenuto in un registro in virgola mobile dell'MC68881, o in un registro di dati dell'MC68020 per valori interi, o in una locazione della memoria. In quest'ultimo caso, esso può essere indirizzato mediante qualunque modo d'indirizzamento di memoria ammesso dall'MC67020. Un operando di destinazione può essere memorizzato in un registro in virgola mobile, in un registro di dati dell'MC68020 o nella memoria. La destinazione nella memoria può essere indirizzata da qualsiasi modalità d'indirizzamento di memoria dell'MC68020, tranne quello relativo al PC. Una tipica istruzione FMOVE potrebbe essere la seguente:

```
FMOVE.S    (A0)+,FP0
```

che trasferisce un valore in singola precisione dalla locazione di memoria indirizzata da A0 al registro in virgola mobile FP0.

L'istruzione FMOVECR (*Floating point MOVE Constant ROM:* trasferisci ROM di costanti in virgola mobile) carica una costante in un registro in virgola mobile. Tale costante è definita da un offset da $00 a $3F nella ROM. Come esempio, l'istruzione

```
FMOVECR.X      #$00,FP0
```

carica FP0 col valore di π come una costante in precisione estesa. Gli altri effetti e valori sono elencati nell'*MC68881/MC68882 User's Manual*.

La Tab. 12.5 mostra la sintassi di un'istruzione diadica generale ed elenca le istruzioni di questo tipo. Sono possibili operazioni di addizione, confronto, divisione, moltiplicazione e sottrazione di valori in virgola mobile, come pure alcune operazioni speciali. Per esse, l'operando di sorgente può essere situato in un registro in virgola mobile, in un registro di dati dell'MC68020 o nella memoria.

Tab. 12.5 *Istruzioni di trasferimento dei dati (MC68881/MC68882).*

(a) Formato dell'operazione.

Istruzione	Sintassi dell'operando	Formato dell'operando	Operazione
F<opd>	<EA>,FPn	B, W, L, S, D, X, P	FPn <funzione> sorgente → FPn
	FPm,FPn	X	

Nota: <opd> è uno qualunque degli specificatori di operazione diadica.

(b) Operazioni diadiche

Istruzione	Funzione
FADD	Somma
FCMP	Confronto
FDIV	Divisione
FMOD	Resto di modulo
FMUL	Moltiplicazione
FREM	Resto IEEE
FSCALE	Esponente di scala
FSGLDIV	Divisione in singola precisione
FSGLMUL	Moltiplicazione in singola precisione
FSUB	Sottrazione

Le operazioni monadiche eseguite dal coprocessore MC68881 sono elencate nella Tab. 12.6. Queste operazioni matematiche agiscono su un operando di sorgente e calcolano il valore della funzione selezionata. Tali funzioni sono tipicamente disponibili come parte di una libreria di subroutine matematiche per linguaggi ad alto livello o come chiamate di funzioni in linguaggi quali il FORTRAN. La funzione monadica duale FSINCOS nella Tab. 12.6 riporta il valore del seno e del coseno dell'operando di sorgente, per ottenere un risparmio di tempo quando si devono programmare operazioni come le trasformate di Fourier.

12.3.2 Applicazioni del coprocessore MC68881

L'aggiunta della capacità di elaborazione in virgola mobile ad un computer estende notevolmente la gamma di applicazioni per il sistema. Praticamente ogni area della scienza e della tecnica richiede la grande precisione e l'intervallo numerico dei calcoli in virgola mobile. Come coprocessore, l'MC68881 rende disponibile la capacità dell'aritmetica in virgola mobile ai sistemi basati sull'MC68020, estendendo efficacemente l'insieme di istruzioni della CPU per includere le istruzioni in virgola mobile dell'MC68881. I programmi creati da queste istruzioni possono essere progettati per risolvere quasi ogni problema matematico passibile di soluzione numerica. Si potrebbero specificare varie categorie dei problemi matematici da risolvere usando l'MC68881, ad esempio:

(a) Aritmetica in virgola mobile

(b) Valutazione di funzioni

(c) Metodi numerici

Tab. 12.6 *Istruzioni monadiche (MC68881/MC68882).*

(a) Formato dell'operazione.

Istruzione	**Sintassi dell'operando**	**Formato dell'operando**	**Operazione**
F<opm>	<EA>,FPn	B, W, L, S, D, X, P	Sorgente → funzione → FPn
	FPm,FPn	X	
	FPn	X	FPn → funzione → FPn

Nota: <opm> è uno qualunque degli specificatori di operazione monadica.

Tab. 12.6 *(continuazione)*

(b) Operazioni monadiche.

Istruzione	Funzione
FABS	Valore assoluto
FACOS	Arcocoseno
FASIN	Arcoseno
FATAN	Arcotangente
FATANH	Arcotangente iperbolica
FCOS	Coseno
FCOSH	Coseno iperbolico
FETOX	e^x
FETOXM1	$e^x - 1$
FGETEXP	Estrazione di esponente
FGETMAN	Estrazione di mantissa
FINT	Estrazione di parte intera
FINTRZ	Estrazione di parte intera, arrotondata a zero
FLOGN	$\ln(x)$
FLOGNP1	$\ln(x + 1)$
FLOG10	$\log_{10}(x)$
FLOG2	$\log_2(x)$
FNEG	Negazione
FSIN	Seno
FSINH	Seno iperbolico
FSQRT	Radice quadrata
FTAN	Tangente
FTANH	Tangente iperbolica
FTENTOX	10^x
FTWOTOX	2^x

(c) Formato dell'operazione monadica duale.

Istruzione	Sintassi dell'operando	Formato dell'operando	Operazione
FSINCOS	<EA>,FPc:FPs	B, W, L, S, D, X, P	SIN(sorgente) → FPs;
	FPm,FPc:FPs	X	COS(sorgente) →1 FPc

L'aritmetica in virgola mobile viene eseguita dalle istruzioni diadiche per effettuare l'addizione, la divisione, la moltiplicazione e la sottrazione. La valutazione delle funzioni include l'impiego delle istruzioni monadiche dell'MC68881 quando la funzione da valutare è una delle funzioni del coprocessore, quali seno e coseno. Funzioni molto più complesse possono essere approssimate come una serie o somma delle funzioni monadiche fornite dall'MC68881. Per risolvere problemi più complicati, si può ricorrere alle tecniche dei metodi numerici. Questi metodi per approssimare funzioni, risolvere equazioni differenziali, o eseguire l'analisi di Fourier combinano di solito le operazioni matematiche fondamentali in algoritmi adatti al problema specifico. Vari riferimenti bibliografici relativi a questo capitolo ed elencati nell'app. E alla fine del libro trattano più dettagliatamente le applicazioni della virgola mobile ed i metodi numerici.

Esempio 12-3

La Fig. 12.13 è un listato in linguaggio assembler di una subroutine che impiega un certo numero di istruzioni in virgola mobile dell'MC68881.[9] Tale subroutine calcola la serie di Fourier di una forma d'onda triangolare, mediante l'equazione:

$$F(\omega t) = \frac{8k_1}{\pi^2}\left(\sum_{i=0}^{N-1} (-1)^i \frac{1}{(2i+1)^2} \sin[(2i+1)\omega t]\right) + k_2$$

in cui l'argomento ωt assume i valori da 0° a 359° con incrementi di un grado. Gli argomenti passati alla subroutine sono il valore massimo della forma d'onda k_1, l'offset da zero k_2, il numero di termini N da calcolare nella somma, e l'indirizzo iniziale in A1 dell'array di valori che rappresentano $F(\omega t)$.

Soltanto i termini dispari di seno sono calcolati, poiché si presuppone che la forma d'onda triangolare sia una funzione dispari rispetto all'asse $\omega t = 0$.[10] Il calcolo di i + 1 come indice dei termini seleziona soltanto i termini dispari. Per ciascun valore di ωt = 0, 1, 2, ..., 359, determinato dal ciclo che inizia dall'etichetta NEXTDEG, sono calcolati N termini. Il ciclo più interno, che inizia dall'etichetta NEXTTRM, calcola il valore della funzione in un unico punto ωt. Tale valore è convertito in un intero e l'offset k_2 viene aggiunto prima che il valore sia registrrato nella memoria come indirizzato da (A1) usando l'indirizzamento con postincremento.[11] Quando sono stati calcolati 360 valori, la subroutine restituisce il controllo al programma chiamante.

---

[9] L'autore desidera ringraziare il Prof. Jim Harde della Mississippi State University per i programmi di esempio che utilizzano l'MC68881.

[10] Il programma è stato scritto per illustrare l'impiego dell'MC68881. Si potrebbero impiegare varie altre simmetrie nella forma d'onda triangolare per ridurre la quantità di calcoli, ma esse non sono state sfruttate in questo esempio.

[11] Non è necessario convertire in valori interi le ampiezze delle forme d'onda. Essa viene effettuata qui in modo che le ampiezze della forma d'onda in ciascun punto possano essere utilizzate come valori d'ingresso per un convertitore digitale/analogico che crea una forma d'onda analogica delle serie di Fourier.

```
                                1.          TTL       FIGURA 12.13
                                2.          LLEN      100
00010000                        3.          ORG       $10000
                                4.    *
                                5.    *   SUBROUTINE PER IL CALCOLO DELLA SERIE DI FOURIER CON N TERMINI
                                6.    *   DI UNA FORMA D'ONDA TRIANGOLARE.
                                7.    *   I VALORI DELLA FUNZIONE SON CALCOLATI, GRADO PER GRADO,
                                8.    *   DA 0 A 359 GRADI E MEMORIZZATI COME INTERI.
                                9.    *
                                10.   *       INPUT:     (D0.W) = K1: VALORE MASSIMO
                                11.   *                               DELLA FORMA D'ONDA
                                12.   *                  (D1.W) = K2: OFFSET
                                13.   *                  (D2.W) = N:  NUMERO DI TERMINI
                                14.   *                  (A1)   = INDIRIZZO INIZIALE DELL'ARRAY
                                15.   *                             DI RISULTATI
                                16.   *
                                17.   *       OUTPUT: ARRAY DI 360 VALORI DELLA FUNZIONE
                                18.   *
                                19.   *   COSTANTI DA ROM DI MC68881
                                20.   *
  # 00000000                    21.   PI      EQU       $00             ;OFFSET IN ROM
  # 0000000F                    22.   ZERO    EQU       $0F
                                23.   *
00010000  F227 E0FF             24.   FTRIAG  FMOVEM.X  FP0-FP7,-(A7)   ;SALVA I REGISTRI
00010004  48E7 FE40             25.           MOVEM.L   A1/D0-D6,-(A7)
                                26.   *
                                27.   *   CALCOLA IL COEFFICIENTE 8K1/(PI)**2
                                28.   *
00010008  F200 5C80             29.           FMOVECR.X #PI,FP1         ;PI
0001000C  F200 04A7             30.           FSGLMUL   FP1             ;PI**2
                                31.   *
00010010  F200 5100             32.           FMOVE.W   D0,FP2          ;CALCOLA IL COEFFICIENTE
00010014  F23C 4927 4002        33.           FSGLMUL.X #8,FP2
          0000 8000 0000
          0000 0000
00010024  F200 0524             34.           FSGLDIV.X FP1,FP2         ;8*K1/(PI**2)
                                35.   *
00010028  F200 5D80             36.           FMOVECR.X #PI,FP3
0001002C  F23C 49A4 4006        37.           FSGLDIV.X #180,FP3        ;1 GRADO;PI/180
          0000 B400 0000
          0000 0000
```

Fig. 12.13 *Esempio di coprocessore in virgola mobile. (Esempio 12-3)*

ESERCIZI

12.3.1

Si rappresenti +1.0 nella memoria per ciascun formato dell'MC68881.

(a) Virgola mobile in singola precisione
(b) Virgola mobile in doppia precisione
(c) Virgola mobile in precisione estesa
(d) Decimale impaccato

12.3.2

Si scriva un programma per il coprocessore MC68881 in linguaggio assembler per sommare una serie nella forma:

$$F(x) = a_0 + a_1x^1 + a_2x^2 + \ldots + a_Nx^N$$

dove i coefficienti ed il valore della funzione devono essere numeri in virgola mobile in singola precisione. Si provi il programma leggendo dalla memoria il valore x ed i coefficienti per la serie:

$$e^x \approx 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots + \frac{x^n}{n!} + \ldots$$

```
                       38.  *
                       39.  *   CALCOLA LA FUNZIONE PER CIASCUNO DEI 360 GRADI
                       40.  *
0001003C  4243         41.          CLR.W     D3            ;INDICE DI GRADI
0001003E  F200 5F8F    42.  NEXTDEG FMOVECR.X #ZERO,FP7     ;AZZERA ACCUMULATORE
00010042  4244         43.          CLR.W     D4            ;INDICE I DEI TERMINI
00010044  F200 0E00    44.          FMOVE.X   FP3,FP4       ;GRADO SUCCESSIVO (WT)
00010048  F203 5227    45.          FSGLMUL.W D3,FP4        ;  WT
                       46.  *
0001004C  3A04         47.  NEXTTRM MOVE.W    D4,D5         ;CALCOLA (2I+1)
0001004E  E38D         48.          LSL.L     #1,D5
00010050  5285         49.          ADDQ.L    #1,D5         ;(2I+1)
00010052  F200 1280    50.          FMOVE.X   FP4,FP5       ;CALCOLA (2I+1)WT
00010056  F205 52A7    51.          FSGLMUL.W D5,FP5
0001005A  F200 170E    52.          FSIN.X    FP5,FP6       ;SIN((2I+1)WT)
                       53.  *
                       54.  *       TERMINE I-ESIMO
                       55.  *
0001005E  CAC5         56.          MULU.W    D5,D5         ;(2I+1)**2
00010060  F205 4324    57.          FSGLDIV.L D5,FP6        ;SIN()/(2I+1)**2
                       58.  *
                       59.  *   CALCOLA IL SEGNO DEL TERMINE
                       60.  *
00010064  0804 0000    61.          BTST      #0,D4         ;SE INDICE E' PARI
00010068  6700 000A    62.          BEQ       EVEN          ;  ALLORA SEGNO = +1
0001006C  F200 1BA8    63.  ODD     FSUB.X    FP6,FP7       ;  ALTRIMENTI SEGNO = -1
00010070  6000 0006    64.          BRA       CONT
00010074  F200 1BA2    65.  EVEN    FADD.X    FP6,FP7
                       66.  *
                       67.  *       N TERMINI?
                       68.  *
00010078  5244         69.  CONT    ADDQ.W    #1,D4         ;SE N TERMINI
0001007A  B444         70.          CMP.W     D4,D2         ;  MEMORIZZA I RISULTATI
0001007C  62 CE        71.          BHI       NEXTTRM       ;  PER UN PUNTO
                       72.  *                               ;ALTRIMENTI, CALCOLA IL TERMINE
                       73.  *                               ;  SUCCESSIVO
                       74.  *
0001007E  F200 0BA7    75.          FSGLMUL.X FP2,FP7       ;N TERMINI
00010082  F206 7380    76.          FMOVE.W   FP7,D6        ;CONVERTE A INTERO
00010086  DC41         77.          ADD.W     D1,D6         ;  E SOMMA L'OFFSET
00010088  32C6         78.          MOVE.W    D6,(A1)+      ;MEMORIZZA IL RISULTATO
                       79.  *
                       80.  *       360 GRADI?
                       81.  *
0001008A  5243         82.          ADDQ.W    #1,D3         ;SE 360 GRADI
0001008C  B67C 0168    83.          CMP.W     #360,D3       ;  RITORNA
00010090  65 AC        84.          BCS       NEXTDEG       ;ALTRIMENTI CALCOLA IL VALORE
                       85.  *                               ;  SUCCESSIVO F(WT)
00010092  F21F D0FF    86.          FMOVEM.X  (A7)+,FP0-FP7
00010096  4CDF 027F    87.          MOVEM.L   (A7)+,A1/D0-D6
0001009A  4E75         88.          RTS
                       89.
```

Fig. 12.13 *Esempio di coprocessore in virgola mobile (continuazione)*

in cui $n! = 1 \cdot 2 \cdot 3 \cdots n$ denota il fattoriale di n. Il numero dei termini N è una variabile nell'intervallo da $N = 3$ a $N = 10$.

12.3.3

Si selezioni uno dei seguenti problemi e si scriva una subroutine dell'MC68881 per calcolare i risultati.

(a) Conversione di coordinate da polari a cartesiane.
(b) Calcolo delle radici dell'equazione quadratica:

$$Ax^2 + Bx + C = 0$$

Come sono trattate le radici complesse?

(c) Calcolo della trasformata discreta di Fourier usando un algoritmo di trasformata veloce di Fourier (*Fast Fourier Transform:* FFT).

12.4 L'UNITA' DI GESTIONE DELLA MEMORIA IMPAGINATA MC68851

L'unità di gestione della memoria impaginata (*Paged Memory Management Unit:* PMMU) può essere incorporata come un coprocessore nei sistemi basati sull'MC68020 per fornire le capacità di rappresentazione della memoria e di gestione della memoria. Operando sotto il controllo del software di sistema, l'MC68851 attua il metodo di rappresentazione o "mappatura" della memoria (*memory mapping*) di un sistema operativo, eseguendo la conversione dagli indirizzi logici generati da un programma agli indirizzi fisici nella memoria. Il meccanismo di protezione della PMMU consente di proteggere regioni selezionate della memoria principale dall'accesso da parte di un programma in esecuzione. Questo accesso può essere limitato in base alla modalità di supervisore o di utente del programma. Inoltre, l'intervallo d'indirizzamento valido ammesso per un programma può essere confinato ad un intervallo predeterminato. L'MC68851 dispone inoltre delle istruzioni di Breakpoint (BKPT) e di chiamata del modulo (*CALL Module:* CALLM) del processore MC68020.

Questo paragrafo descrive le capacità fondamentali dell'MC68851 e presenta le applicazioni possibili della PMMU. Sono posti in evidenza gli aspetti di programmazione di questo coprocessore, ma si rimanda il lettore all'*MC68851 Paged Memory Management Unit User's Manual* per ulteriori dettagli concernenti l'attività e la programmazione della PMMU. Le capacità di rappresentazione della memoria e di protezione possedute dall'MC68851 sono simili a quelle fornite dall'unità di gestione della memoria sul chip del microprocessore MC68030 che sarà descritto nel cap. 15. In questo capitolo sono descritte le differenze tra le due unità di gestione della memoria. Le forme del linguaggio assembler per le istruzioni dell'MC68851 sono elencate nell'app. C.

La Fig. 12.14 illustra la connessione tipica della PMMU in un sistema di computer. Gli indirizzi logici generati dalla CPU mentre essa esegue le istruzioni di un programma sono "sorvegliati" dall'MC68851. Ogni indirizzo logico viene controllato in base al suo privilegio o intervallo e, se ritenuto valido, viene convertito in un indirizzo fisico che rappresenta una locazione nella memoria o la locazione di un dispositivo periferico. Quindi tutti gli accessi alla memoria o ai dispositivi periferici per operazioni di lettura o di scrittura sono convalidati e controllati dalla MC68851 quando è installata in un sistema come mostrato nella Fig. 12.14. In definitiva, la relazione esatta tra gli indirizzi fisici e quelli logici ed il tipo di protezione della memoria offerta dalla PMMU sono determinati dalla routine del sistema operativo che programma la PMMU.

Considerazioni sul sistema operativo. Un sistema operativo dev'essere presente per svolgere le funzioni di gestione della memoria per un sistema di computer. Nel caso più semplice di un sistema monoutente in cui un solo programma di utente alla volta risiede nella memoria, il sistema operativo e l'unità di gestione della memoria servono a proteggere lo spazio di memoria allocato al sistema operativo contro l'accesso da parte del programma in modo utente. Per un sistema

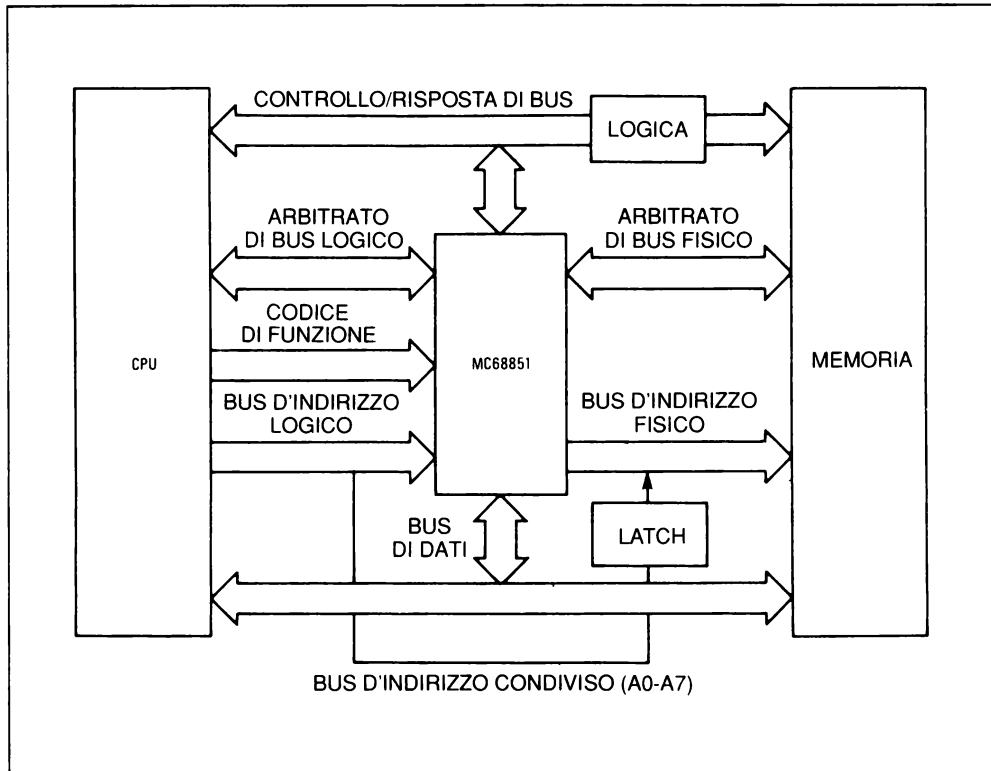



Fig. 12.14 *La PMMU MC68851 in un sistema basato sull'MC68020.*

multiprogrammato o multiutente, in cui diversi programmi di utente risiedono simultaneamente nella memoria e condividono la CPU, l'area del sistema operativo nella memoria dev'essere protetta contro l'accesso da parte di qualunque utente, ed ogni area di utente dev'essere protetta contro altri utenti. Tali protezioni sono facilmente fornite dall'MC68851.

I sistemi operativi multiprogrammati più avanzati consentono che una porzione di programma di utente risieda nella memoria per l'esecuzione, mentre altre sezioni del programma sono contenute in un dispositivo di memoria secondaria quale un'unità di memorizzazione a disco. Quando si rendono necessarie le parti del programma registrate nella memoria secondaria, esse vengono caricate nella memoria principale dal sistema operativo. Gli indirizzi del programma non coincidono necessariamente con gli indirizzi fisici a cui il programma ed i suoi dati vengono caricati. Mentre il programma viene eseguito, l'MC68851 converte gli indirizzi *logici* o di programma negli indirizzi fisici appropriati, in accordo con la tabella di conversione degli indirizzi creata dal sistema operativo. Nei sistemi basati sull'MC68020 che impiegano l'MC68851, il programma sulla memoria secondaria è suddiviso in *pagine* a cui il sistema operativo fa riferimento. Nella memoria, tali pagine possono risiedere in aree fisiche della medesima dimensione in byte. Poiché gli indirizzi logici

o di programma vengono convertiti mentre il programma viene eseguito, le pagine del programma possono risiedere ovunque nella memoria. Una pagina di istruzioni può quindi essere eseguita da qualsiasi punto nella memoria, anche se il programma non è indipendente dalla posizione, in base al la definizione fornita nel par. 9.2.

Un sistema operativo può anche eseguire uno "scambio" (*swapping*) di pagine se il programma necessita di più area di memoria di quanta non ne sia disponibile. Se una pagina nella memoria viene sostituita conformemente a qualche algoritmo definito dal sistema operativo, allora essa dev'essere scritta di nuovo sull'unità a disco, qualora sia stata modificata mentre si trovava nella memoria.[12] Nei sistemi basati sull'MC68020, queste considerazioni dovrebbero valere soltanto per pagine di valori di dati, poiché i programmi non dovrebbero scrivere in aree di memoria contenenti soltanto istruzioni. In ogni caso, l'MC68851 tiene nota delle pagine che sono state modificate, cosicché un sistema operativo in grado di scambiare le pagine non avrà bisogno di trasferire nella memoria secondaria una pagina rimasta invariata nella memoria, poiché l'unità di memorizzazione contiene una versione valida (cioè, una copia identica) della pagina non modificata. Complessivamente, il tempo di esecuzione del programma si riduce quando si evitano operazioni di scrittura non necessarie nella memoria secondaria.

12.4.1 Rappresentazione della memoria da parte dell'MC68851

L'MC68851 può leggere l'indirizzo generato dalla CPU durante l'esecuzione del programma ed utilizzarlo per calcolare un indirizzo nella memoria. Per distinguere tra i due indirizzi, quello generato dalla CPU è definito indirizzo *logico*. Esso rappresenta l'indirizzo associato con un programma in corso di esecuzione per prelevare un'istruzione o per leggere o scrivere un valore in una locazione di memoria. Gli indirizzi nella memoria principale sono denominati indirizzi *fisici*. In parole semplici, l'MC68851 converte o "trasforma" gli indirizzi logici in indirizzi fisici, in accordo con le tabelle di conversione registrate nella memoria o in un dispositivo di memoria secondaria come un'unità a disco. In questo paragrafo, si supporrà che tali tabelle di conversione siano contenute nella memoria principale e che pertanto siano accessibili da parte dell'MC68851 senza l'intervento di un sistema operativo. Inoltre, si farà l'ipotesi che ciascun indirizzo logico corrisponda ad un indirizzo fisico (cioè, lo spazio di indirizzi logici e lo spazio di indirizzi fisici hanno la stessa dimensione). Queste ipotesi consentono di focalizzare la discussione sull'attività iniziale della PMMU e non sullo spazio di allocazione della memoria o sullo spazio di memorizzazione secondaria da parte del software di sistema.

Il coprocessore MC68851 è definito come un'unità di gestione della memoria *impaginata* poiché gli spazi degli indirizzi sia logici che fisici sono considerati suddivisi in pagine di dimensione fissa. Per ogni pagina logica, consistente di 2^m byte

[12] Le tecniche con cui il sistema operativo effettua lo scambio di pagine sono descritte in vari riferimenti bibliografici relativi a questo capitolo, riportati nell'app. E alla fine del libro.

di istruzioni o di dati, c'è una pagina corrispondente nella memoria fisica della medesima dimensione, denominata *frame di pagina.*[13]

La Fig. 12.15(a) illustra la relazione concettuale tra lo spazio di indirizzi logici contenente uno o più programmi e la memoria fisica suddivisa in frame di pagina. Un programma in forma di codice-oggetto, contenente gli indirizzi logici assegnati dal programmatore o da un compilatore o dall'editor di collegamento, è contenuto in una memoria "virtuale" secondaria, che nella maggior parte dei casi è un'unità di memorizzazione su disco. Gli indirizzi fisici vengono assegnati dalla PMMU usando la tabella di conversione contenuta nella memoria mentre il programma è in corso di esecuzione. Dopo che il programma è stato inizialmente caricato dal sistema operativo in uno o più frame di pagina, il controllo viene passato al programma, inserendo nel contatore di programma l'indirizzo logico iniziale del programma stesso. Il metodo adottato per trasferire il controllo nei sistemi basati sull'MC68020 che usano l'istruzione RTE è stato discusso nel par. 10.2.

Mentre il programma viene eseguito, la CPU preleva le istruzioni come se stesse leggendo dallo spazio di indirizzi logici. La PMMU converte questi indirizzi e fa sì che l'istruzione nella memoria sia passata ordinatamente alla CPU. La Fig. 12.15(a) mostra che l'ubicazione effettiva delle pagine associate col programma non deve necessariamente occupare frame di pagina contigui nella memoria. Quindi, mediante questo metodo, un sistema operativo può assegnare pagine di programma alla memoria in qualsiasi maniera conveniente.

La Fig. 12.15(b) illustra il meccanismo di conversione fondamentale da indirizzi logici ad indirizzi fisici. Si noti che un indirizzo logico è suddiviso in due campi. Quando la CPU emette un indirizzo di n bit sulle proprie linee di segnali d'indirizzo, la PMMU impiega gli $n - m$ bit superiori per determinare un valore nella tabella di conversione. La locazione effettiva nella tabella suddetta è:

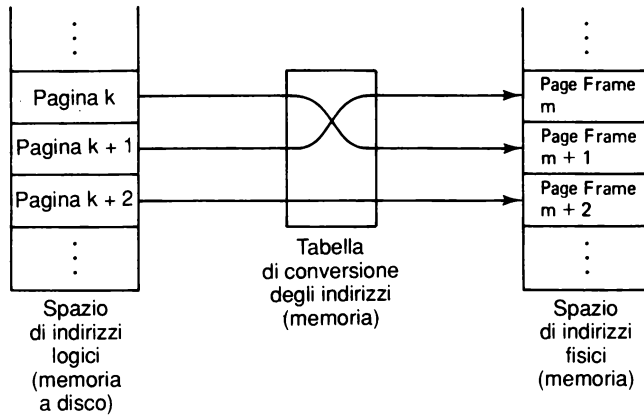
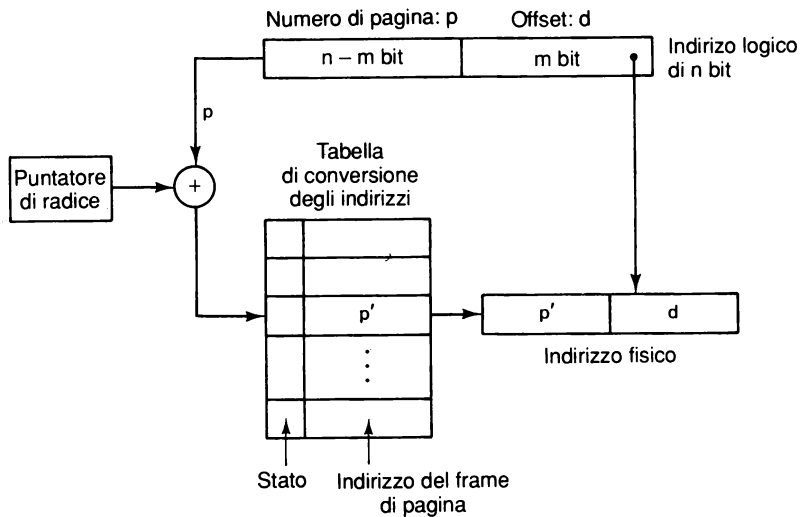
(puntatore di radice) + p * (dimensione delle entrate, in byte)

dove il puntatore di radice indirizza la base della tabella, mentre il valore di p seleziona il numero dell'entrata (elemento indirizzabile) nella tabella. Lo spostamento in byte dell'entrata dall'indirizzo di base è p volte la dimensione delle entrate. Questa tabella ha 2^{n-m} entrate nella Fig. 12.15(b). Questo metodo è noto come *rappresentazione diretta* (*direct mapping*) poiché la tabella di conversione contiene un'entrata per ogni pagina nello spazio di indirizzi logici. L'MC68851 consente la rappresentazione diretta, oltre a metodi di rappresentazione più complessi.

L'entrata per ciascuna pagina fisica nella tabella di conversione rappresentata direttamente contiene l'indirizzo del frame di pagina, che è l'indirizzo base del frame di pagina nella memoria. L'indirizzo fisico usato per accedere alla memoria è la concatenazione di questo indirizzo di frame di pagina e degli m bit inferiori dell'indirizzo logico. Nella Fig. 12.15(b), l'indirizzo p' seleziona il frame di pagina, mentre l'offset d seleziona una locazione specifica nel frame di pagina. La dimensione della pagina è di 2^m byte.

[13] Altre tecniche di rappresentazione della memoria, come la segmentazione, sono discusse in vari riferimenti bibliografici relativi a questo capitolo, nell'app. E.

Pagina k
Pagina k + 1
Pagina k + 2
Spazio di indirizzi logici (memoria a disco)
Tabella di conversione degli indirizzi (memoria)
Page Frame m
Page Frame m + 1
Page Frame m + 2
Spazio di indirizzi fisici (memoria)

(a)

Numero di pagina: p
Offset: d
n – m bit
m bit
Indirizo logico di n bit
p
Puntatore di radice
+
Tabella di conversione degli indirizzi
p'
p'
d
Indirizzo fisico
Stato
Indirizzo del frame di pagina

Note:
(1) La tabella di conversione ha 2^{n-m} entrate nella memoria;
64 entrate possono essere contenute nella memoria cache della CPU.
(2) La dimensione della pagina è di 2^m byte.
(3) p' è l'indirizzo di base del frame di pagina selezionato.

(b)

Fig. 12.15 *(a) Il concetto di spazio logico confrontato con quello di spazio di indirizzi; (b) il concetto generale di conversione di indirizzo.*

La tabella di conversione degli indirizzi contiene anche informazioni di stato. Queste informazioni determinano gli attributi e la protezione per il frame di pagina a cui accedere. Il significato delle varie condizioni di stato è descritto nell'*MC68851 User's Manual.*

Per ridurre il tempo di conversione, l'MC68851 dispone di un cache di conversione dell'indirizzo, che contiene le 64 conversioni più recenti da indirizzo logico a indirizzo fisico. Quando un'entrata viene trovata nel cache, l'MC68851 non ha bisogno di accedere alla tabella di conversione dell'indirizzo nella memoria.

L'insieme di registri dell'MC68851. L'insieme di registri dell'MC68851 contiene vari registri programmabili. Sono disponibili tre cosiddetti registri di puntatore di radice per puntare alle tabelle di conversione nella memoria. Essi contengono lo stato ed altre informazioni riguardanti le tabelle di conversione, come pure gli indirizzi di base. Questi tre registri consentono di distinguere le tabelle di conversione del modo di supervisore, le tabelle di accesso diretto alla memoria e le altre tabelle di conversione. Un registro di controllo della conversione, di 32 bit, è impiegato per inizializzare la PMMU e definire i campi di un indirizzo logico, al fine di determinare la dimensione della pagina ed altri attributi. Un bit in questo registro può essere posto a {0} per disabilitare la PMMU. Quando la PMMU è disabilitata, non ha luogo alcuna conversione di indirizzi, per cui non viene riconosciuta la distinzione tra gli indirizzi fisici e quelli logici. La PMMU viene solitamente abilitata durante la sequenza d'inizializzazione da un sistema operativo che necessita di una conversione di indirizzi.

Sono disponibili due registri di stato per consentire al software di sistema di determinare lo stato del registro di cache dell'MC68851 e le condizioni che risultano quando un indirizzo viene convertito. La memoria cache della PMMU può contenere fino a 64 coppie di indirizzi logico-fisici che sono stati convertiti. Quando una coppia è presente nella memoria cache della PMMU, l'MC68851 non ha bisogno di eseguire una ricerca nella tabella di conversione contenuta nella memoria principale. Per ulteriori dettagli concernenti il formato esatto del contenuto dei registri della PMMU che controllano il processo di conversione, si rimanda il lettore all'*MC68851 User's Manual.*

L'insieme di istruzioni dell'MC68851. L'insieme di istruzioni per l'MC68851 è elencato nell'App. C. Le istruzioni consentono ad un programma nel modo di supervisore di controllare l'attività della PMMU e della sua memoria cache, oltre a disporre di altre funzioni. L'istruzione primaria per il controllo della PMMU è PMOVE. Essa è utilizzata per trasferire i dati da o verso un registro dell'MC68851, impiegando le modalità d'indirizzamento della CPU, quando il trasferimento riguarda i registri della CPU o la memoria. Ad esempio, l'istruzione

```
PMOVE    (A1),CRP
```

trasferisce una doppia longword puntata da A1 dalla memoria al registro di 64 bit del puntatore di radice della CPU (*CPU Pointer Register:* CPR) nell'MC68851.

Tale registro individua la tabella di conversione degli indirizzi nella memoria. Altre istruzioni dell'MC68851 sono descritte in dettaglio nell'*MC688512 User's Manual.*

12.4.2 Protezione della memoria mediante l'MC68851

La PMMU MC68851 è impiegata in molti sistemi per fornire la protezione allo spazio di memoria di un programma contro l'accesso non autorizzato da parte di un altro programma. I meccanismi di protezione più importanti consentono ad un sistema operativo di separare gli spazi di indirizzi di supervisore e di utente e di limitare l'intervallo d'indirizzamento di un programma. Inoltre, frame di pagina selezionati possono essere dichiarati come aree di memoria a sola lettura. Se un programma tentasse di violare le limitazioni imposte da un sistema operativo, il tipo di violazione sarebbe indicato nel registro di stato dell'MC68851. Si rimanda il lettore all'*MC68851 User's Manual* per ulteriori dettagli in merito alle protezioni consentite in un sistema con capacità di gestione della memoria.

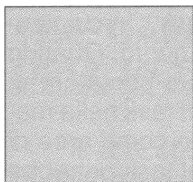
12.4.3 Altre caratteristiche della PMMU MC68851

La PMMU MC68851 dispone dei registri per consentire ad un programma di utilizzare le istruzioni di breakpoint (BKPT) e di chiamata di modulo (*CALL Module:* CALLM) dell'MC68020. L'istruzione BKPT serve a facilitare il debugging di un programma, come spiegato nel par. 11.3. CALLM consente ad un programma di chiamare una routine in un altro programma quando questa è progettata come un'unità indipendente, denominata *modulo.* Si può associare un livello di privilegio a ciascun modulo, cosicché l'accesso al modulo può essere ristretto. Queste caratteristiche sono descritte in dettaglio nell'*MC68851 User's Manual.* Le istruzioni sono anche elencate nell'app. D di questo libro.

ESERCIZI

12.4.1 Si descrivano i vari modi in cui un indirizzo logico viene assegnato da un programmatore o dal software di sistema. Si discutano i vantaggi e gli svantaggi di ciascun metodo di assegnazione degli indirizzi.

12.4.2 Si considerino indirizzi logici vincolati ad una lunghezza di 27 bit, con i 12 bit inferiori usati come offset in un frame di pagina. Il registro del puntatore di radice contenga il valore $10000, mentre la memoria corrispondente contenga i seguenti indirizzi:

Locazione di memoria	Indirizzo di frame di pagina
$10000	$745A7
$10004	$36271
$10008	$F01E2

(a) Quale indirizzo fisico corrisponde all'indirizzo logico \$123?

(b) Quale indirizzo fisico corrisponde all'indirizzo logico \$1CE5?

(c) Qual è la dimensione in byte della tabella di conversione degli indirizzi?

12.5 CONSIDERAZIONI SULL'IMPIEGO DELLO STACK PRINCIPALE DA PARTE DI UN SISTEMA OPERATIVO

Un sistema operativo in multiprogrammazione seleziona un programma o un task dopo l'altro per l'esecuzione. I task possono essere pianificati per avviare l'esecuzione ed essere fermati in base al verificarsi di qualche evento nel sistema. Per esempio, in un sistema in time-sharing (condivisione di tempo), ad un task è assegnato un intervallo di tempo specifico per l'esecuzione. Se esso non giunge a termine nel tempo assegnato, allora viene temporaneamente sospeso ed il controllo della CPU viene assegnato ad un altro task. Infine il task sospeso riacquisterà il controllo della CPU per proseguire la propria esecuzione. Questo trasferimento del controllo tra i task è talvolta indicato come *commutazione del contesto* (*context switching*). In un sistema in time-sharing, il verificarsi di un'interruzione da un temporizzatore hardware nel sistema causa una commutazione di contesto. Dopo una o più di queste interruzioni che definiscono la durata dell'intervallo disponibile per un task, il sistema operativo in time-sharing passa il controllo dal task corrente a quello pianificato per essere eseguito successivamente. Sistemi operativi di altro tipo possono trasferire il controllo e quindi effettuare la commutazione del contesto in base al metodo di priorità o alle interruzioni da dispositivi di I/O e a dispositivi speciali nel sistema. In ogni caso, in sistemi basati sull'MC68020, le informazioni in merito al task sospeso devono essere salvate nello stack di sistema per poi essere ripristinate quando il task sospeso riacquista il controllo della CPU. Nei sistemi operativi in multiprogrammazione (multitasking), il *puntatore di stack principale* dell'MC68020 può essere utilizzato per puntare all'area di stack per i task il cui contesto viene salvato mentre sono temporaneamente in sospeso.

Il *contesto* per un task che dev'essere salvato è definito come il contenuto di qualsiasi registro che sta utilizzando, insieme coi valori del registro di stato e del contatore di programma, nonché altre informazioni che dipendono dalla natura stessa del task. In alcuni casi, potrebbe essere necessario salvare sezioni di programmi o di dati di un task su un'unità di memoria secondaria. Il salvataggio ed il ripristino delle informazioni necessarie si ottiene mediante l'appropriata routine del sistema operativo per la gestione delle eccezioni nei computer basati sul processore MC68020.

Poiché molti task indipendenti possono esistere contemporaneamente nella memoria, il sistema operativo può assegnare a ciascun task un'area specifica

della memoria, che viene denotata come "blocco di controllo del task", in cui vengono salvati i valori dei registri di un task ed altre informazioni in merito. Quando un particolare task viene sospeso, il sistema operativo punta il suo puntatore di stack di sistema alla locazione appropriata nel blocco di controllo del task nella memoria ed inserisce le informazioni richieste nelle locazioni riservate come aree di stack per questo task. Un'area di stack in un blocco di controllo del task potrebbe essere creata per ciascun task del sistema. L'accesso a questi stack sarebbe limitato alla routine del sistema operativo che salva e ripristina i valori dall'area di stack per un certo task.

Per facilitare il progetto di sistemi operativi che attuano la commutazione dei task, l'MC68020 dispone di due puntatori di stack per un programma operante nel modo di supervisore. L'impiego di due stack di supervisore è facoltativo, ma se è richiesto da un sistema operativo, gli stack distinti possono essere controllati da un puntatore di stack principale (*Master Stack Pointer:* MSP) e da un puntatore di stack d'interruzione (*Interrupt Stack Pointer:* ISP). Lo stack principale viene selezionato quando il sistema operativo pone a {1} il bit M nel registro di stato (SR[12]), come descritto nel par. 10.2.[14] Quando viene utilizzato il puntatore di stack principale, tutte eccezioni che non siano interruzioni sono elaborate usando il puntatore di stack principale. Comunque, un frame di stack d'interruzione è creato sullo stack principale come descritto nell'esercizio 12.5.1. Le interruzioni vengono elaborate usando il puntatore di stack d'interruzione per salvare le informazioni sullo stack d'interruzione, come è stato descritto nel par. 11.6.

ESERCIZIO

12.5.1

Si definiscano i valori assunti dal bit S e dal bit M del registro di stato salvato quando s'impiega lo stack principale e si verifica un'interruzione per ciascuno dei seguenti casi:

(a) Il valore di (SR) per uno stack di utente che viene salvato nel blocco di controllo del task.

(b) Il valore di (SR) che viene salvato nello stack d'interruzione.

Quando si verifica un'interruzione, viene creato dapprima un frame di stack di quattro word nello stack principale. Esso contiene la word di formato ed i valori di (PC) e di (SR) per il task. Un secondo frame di stack di quattro word viene creato nello stack d'interruzione. Il secondo frame di stack contiene il medesimo (PC) e l'offset del vettore nella word di formato, ma un diverso numero di formato e (SR). Per lo stack d'interruzione, i bit S e R del registro di stato assumono entrambi il valore {1}, indipendentemente dal modo del programma interrotto. Dopo che l'interruzione è stata gestita, l'istruzione RTE elabora entrambi i frame di stack a partire dallo stack d'interruzione.

[14] Ogni stack può avere un indirizzo di puntatore di stack principale associato con esso. Quando il sistema operativo commuta tra i task per l'esecuzione, l'MSP può essere caricato con un indirizzo diverso per ciascun task.

12.6 CAPACITA' DI MULTIELABORAZIONE DELL'MC68020

Il concetto di multielaborazione in un sistema basato sull'MC68020 è stato introdotto nel par. 2.3. In particolare, la Fig. 2.10 mostrava la possibile configurazione di un sistema multiprocessore in cui vari processori ed altre unità condividono il bus di computer del sistema ed un'area di memoria comune. La Fig. 12.16 illustra un altro esempio di sistema multiprocessore in cui ogni CPU ha la propria area di memoria privata che non è accessibile da parte di altre CPU. Entrambi i processori condividono un'area di memoria comune ed un dispositivo periferico d'ingresso/uscita quale una stampante. L'unità di arbitrato di bus riceve le richieste tramite le linee dei segnali di controllo da una CPU per utilizzare il bus. Una richiesta viene soddisfatta facendo sì che l'altra CPU rilasci il bus dopo il completamento di un ciclo di bus, a meno che non sia attivata la sua linea di segnale RMC. Se il segnale RMC (ciclo di lettura-modifica-scrittura) è emesso da una certa CPU, l'arbitrato di bus non accoglierà la richiesta di un'altra CPU. Questo segnale produce un ciclo di istruzione *indivisibile*. Questo è il metodo con cui una CPU indica che deve avere un accesso esclusivo ad una locazione nella memoria condivisa, come descritto nel par. 12.6.1.

CPU1
Memoria per CPU1
Memoria condivisa
Memoria per CPU2
CPU2
$\overline{RMC}$
$\overline{RMC}$
Unità di arbitrato di bus
Indirizzo, controllo e dati (bus di sistema)
Dispositivo di I/O condiviso

Note:
1. Se una CPU invia il segnale RMC, allora essa ha l'uso esclusivo del bus di sistema fino al termine della sua istruzione corrente.
2. Le istruzioni TAS, CAS e CAS2 causano l'attivazione del segnale RMC finché la loro esecuzione non è stata completata.
3. Il ciclo di lettura-modifica-scrittura della CPU e la linea di segnale RMC sono descritti più in dettaglio nel par. 13.4.

Fig. 12.16 *Esempio di sistema multiprocessore.*

Un programmatore fa sì che una CPU attivi la linea di segnale RMC ed abbia l'accesso esclusivo ad una locazione condivisa della memoria utilizzando una delle istruzioni dell'MC68020 progettate per prevenire l'insorgere di conflitti sul bus di sistema. Queste istruzioni sono TAS (*Test And Set:* esamina e imposta), CAS (*Compare And Swap:* confronta e scambia) e CAS2 (*Compare And Swap 2:* confronta e scambia due volte), descritte nel par. 12.6.2. Lo scopo della protezione di una variabile condivisa nella memoria è quello di garantire che una CPU non legga il valore mentre un'altra CPU sta eseguendo un'istruzione per modificarlo.

12.6.1 Arbitrato di bus nei sistemi multiprocessore

Varie linee di segnale per l'arbitrato di bus sono incluse nel bus di sistema in sistemi basati sull'MC68020. In un sistema multiprocessore, una CPU può assumere il controllo del sistema avanzando una richiesta di bus.[15] Quando tale richiesta viene accolta, la CPU può eseguire un ciclo di bus per leggere o scrivere un valore nella memoria. Poiché la maggior parte delle istruzioni richiede vari cicli di bus, un ciclo di istruzione potrebbe essere interrotto da una richiesta di bus avanzata da un altro processore. Ciò non crea alcun problema se la CPU il cui ciclo di istruzione viene interrotto non sta condividendo locazioni di memoria o dispositivi d'ingresso/uscita con un altro processore. Dopo che la seconda CPU avrà rilasciato il bus, il processore il cui ciclo d'istruzione era stato interrotto potrà continuare la sua operazione e completare l'istruzione in sospeso.[16]

Quando più processori condividono simultaneamente delle risorse come la memoria e i dispositivi d'ingresso/uscita, dev'essere disponibile qualche mezzo per controllare l'accesso a tali risorse. In particolare, certe operazioni programmate devono essere eseguite da una certa CPU senza la possibilità che un altro processore utilizzi la medesima risorsa condivisa durante l'esecuzione della sequenza d'istruzione critica. Per esempio, se un processore sta creando o modificasndo dei valori in una tabella condivisa nella memoria che sarà utilizzata da un secondo processore, è d'importanza capitale che tali valori non vengano letti o utilizzati dal secondo processore finché non sarà stata completata la sequenza di creazione o modifica da parte del primo processore.

Nei sistemi basati sull'MC68020, la tecnica adottata per prevenire l'accesso simultaneo ad una risorsa condivisa da più processori richiede l'impiego di istruzioni che causano un ciclo indivisibile di lettura-modifica-scrittura. Tali istruzioni (TAS, CAS e CAS2) impediscono infatti ad un altro processore d'interrompere i cicli di bus di un processore che stia eseguendo una di tali istruzioni. Pertanto, una

---

[15] La circuiteria di arbitrato di bus, che serve a consentire la condivisione del bus di sistema da parte di più processori e dell'unità di accesso diretto alla memoria, sarà descritta più dettagliatamente nel cap. 13.

[16] L'interruzione menzionata qui si presenta sul bus di sistema, ciclo per ciclo. Essa non dovrebbe essere confusa col riconoscimento di una *interruzione* da parte della CPU, come discusso nel par. 11.6. Un'interruzione può essere elaborata soltanto dopo che un intero ciclo di istruzione è stato completato.

variabile condivisa può essere letta, modificata se necessario, e infine scritta di nuovo nella memoria da un certo processore, senza alcun pericolo che un altro processore legga o modifichi la variabile condivisa prima ancora che il processore corrente abbia completato il suo ciclo d'istruzione. Un siffatto ciclo d'istruzione indivisibile viene indicato a tutti gli altri processori nel sistema quando la CPU che sta eseguendo un'istruzione TAS, CAS o CAS2 esegue un ciclo di lettura-modifica-scrittura ed attiva la sua linea di segnale RMC. Questa ed altre linee di segnale impiegate per consentire l'arbitrato di bus saranno descritte in dettaglio nel par. 13.4.

12.6.2 Le istruzioni TAS, CAS e CAS2

L'MC68020 dispone di due metodi per consentire la sincronizzazione di programmi tra più processori che condividono il medesimo sistema di bus. Nel primo metodo, l'istruzione di "esame e assegnazione" (*Test And Set:* TAS) serve a controllare un bit di flag condiviso, talvolta denominato *semaforo.* La lettura e la modifica del semaforo è permessa ad un solo processore, una volta che questa CPU ha acquisito l'accesso al bus di sistema. Non è consentito alcun altro accesso alla locazione di memoria che contiene il semaforo, a meno che l'istruzione TAS non sia stata completata. L'altro metodo consente di aggiornare uno o due operandi senza la possibilità di un conflitto tra più processori. Le istruzioni di confronto e scambio (*Compare And Swap:* CAS) e di confronto e scambio doppio (*Compare and Swap 2:* CAS2) permettono che uno o due operandi, rispettivamente, siano condivisi tra i processori. L'istruzione TAS è stata presentata nel par. 8.3. Le istruzioni di confronto e scambio sono state discusse brevemente nel par. 9.3.

La Tab. 12.7 riassume le operazioni fondamentali delle istruzioni TAS, CAS e CAS2. I dettagli operativi di queste istruzioni sono descritti nell'app. D di questo libro. Varie applicazioni e diversi "trucchi" di programmazione riguardanti l'impiego di tali istruzioni in sistemi multiprocessore sono descritti in alcuni riferimenti bibliografici relativi a questo capitolo, elencati nell'app. E alla fine del libro. Le istruzioni CAS e CAS2 sono anche descritte in dettaglio nell'app. D dell'*MC68020 User's Manual.*

Tab. 12.7 *(a) L'istruzione TAS.*

Sintassi	Operazione
TAS <EA>	IF (EA)[7:0] = 0 Poni N = {0}, Z = {1} ELSE IF (EA)[7] = {1} Poni N = {1}, Z = {0} Poni (EA)[7] = {1}

Nota: <EA> è definito da un modo d'indirizzamento alterabile di dati; tutti i modi tranne quello diretto di registro e quello relativo al PC.

Tab. 12.7 *(b) L'istruzione CAS.*

Sintassi		Operazione
CAS.<l>	Dc,Du,<EA>	IF (EA) = (Dc) aggiorna: (EA) ← (Du) ELSE Z = {0} e (Dc) ← (EA)

Nota: <EA> è definito da un modo d'indirizzamento alterabile di memoria; tutti i modi tranne quello diretto di registro e quello relativo al PC.

Tab. 12.7 *(c) L'istruzione CAS2.*

Sintassi		Operazione
CAS2.<l1>	Dc1:Dc2,Du1:Du2,(Rn1):(Rn2)	IF (EA1) = (Dc1) AND (EA2) = (Dc2) aggiorna: (EA1) ← (Du1) e (EA2) ← (Du2) ELSE Z = {0} e (Dc1) ← (EA1) e (Dc2) ← (EA2)

Note:
1. <l> = B, W o L; <l1> = W o L.
2. Dc, Dc1 e Dc2 sono registri di "confronto"; uno qualunque di D0-D7.
3. Du, Du1 e Du2 sono registri di "aggiornamento"; uno qualunque di D0-D7.
4. Rn1 e Rn2 contengono gli indirizzi degli operandi esaminati nella memoria (EA1) ed (EA2).
5. I codici di condizione N, Z, V e C sono modificati in accordo coi risultati del confronto. Se i valori da confrontare sono uguali, allora Z = {1}.

ESERCIZI

12.6.1 Si scriva una sequenza di istruzioni che svolgono la medesima funzione della seguente:

```
CAS.B        D2,D0,(A4)
```

se non sono importanti considerazioni di multiprocessore, cioè senza impiegare un ciclo indivisibile di lettura-modifica-scrittura.

12.6.2 Si scriva una sequenza di istruzioni che svolgono la medesima funzione di

```
CAS2.L       D0:D2,D3:D0, (A0),(A1)
```

se le considerazioni di multiprocessore non sono importanti.

12.6.3

In quali condizioni le istruzioni TAS, CAS e CAS2 sarebbero utili per proteggere le variabili condivise in un sistema monoprocessore? [*Suggerimento:* si considerino sistemi multiprogrammati in cui l'esecuzione di vari compiti (*task*) è controllata da interruzioni.]

CAPITOLO 13

INTERFACCIAMENTO E PROGRAMMAZIONE DI CHIP PERIFERICI

In questo capitolo saranno discussi i requisiti d'interfacciamento e la programmazione dei chip periferici per sistemi basati sull'MC68020. Dapprima, nel par. 10.1 saranno definite le caratteristiche generali delle interfacce per descrivere alcuni sistemi tipici. Al giorno d'oggi, questi sistemi sono progettati utilizzando chip periferici per realizzare la maggior parte della circuiteria d'interfaccia. La programmazione di tali chip sarà trattata nel par. 13.2 allorché saranno definiti i loro requisiti speciali. I metodi fondamentali per la programmazione dei dispositivi d'interfacciamento considerati qui sono applicabili a molti chip disponibili in commercio.

Durante la progettazione del sistema, il calcolo dei requisiti di temporizzazione per programmi è talvolta importante per poter effettuare una previsione delle prestazioni del sistema. Il tempo esatto richiesto da qualsiasi istruzione o il tempo per avviare l'elaborazione delle eccezioni può essere determinato mediante i tempi di esecuzione delle istruzioni, pubblicati nell'*User's Manual* della Motorola per l'MC68020. Un certo numero di esempi nel par. 13.3 illustrerà il modo in cui questi tempi possono essere usati per determinare la temporizzazione di sistemi basati sull'MC68020.

La comprensione delle linee di segnale della CPU è necessaria per la progettazione delle interfacce che si connettono al bus di sistema. Le linee di segnale sono utilizzate per i trasferimenti di dati, la protezione e la gestione della memoria, l'elaborazione delle interruzioni, il controllo del bus, e la rivelazione di errori di hardware. Nel par. 13.4 saranno trattate le linee di segnale dal punto di vista del progettista dell'hardware. Nel par. 13.4 sarà anche descritta l'interfaccia di coprocessore dell'MC68020.

Questo capitolo concluderà la descrizione dell'MC68020 sia come una CPU programmabile che come chip di circuito integrato. Nel cap. 14 saranno presentati i sistemi di VMEbus che impiegano l'MC68020 come unità centrale di elaborazione. Il processore MC68030 sarà descritto nel cap. 15, in cui le sue capacità saranno confrontate con quelle dell'MC68020.

13.1 PROGETTAZIONE DELL'INTERFACCIA

La Fig. 13.1 illustra l'organizzazione di un tipico sistema di computer, evidenziando il ruolo svolto dalla circuiteria d'interfacciamento. L'interfaccia connette elettricamente il bus di sistema interno col controllore del dispositivo e risolve le differenze di temporizzazione o di formato tra la CPU e il dispositivo esterno. La CPU, tramite la sua circuiteria logica e le routine di I/O, controlla l'attività del sistema. Le routine di I/O preparano l'interfaccia per i trasferimenti di dati col dispositivo periferico, che di per sé è controllato dal controllore del dispositivo.

Quando il dispositivo è inizializzato e pronto per il trasferimento, esso può svolgere la sua funzione, come la trasmissione di un carattere alla CPU o alla memoria. I trasferimenti alla CPU di solito implicano una richiesta d'interruzione. I trasferimenti ad alta velocità di blocchi di caratteri sono effettuati con richieste di DMA. Se il dispositivo impiega una tecnica di accesso diretto alla memoria (*Direct Memory Access:* DMA), la richiesta di trasferimento passa attraverso l'interfaccia ai circuiti di controllo del bus ed alla memoria.

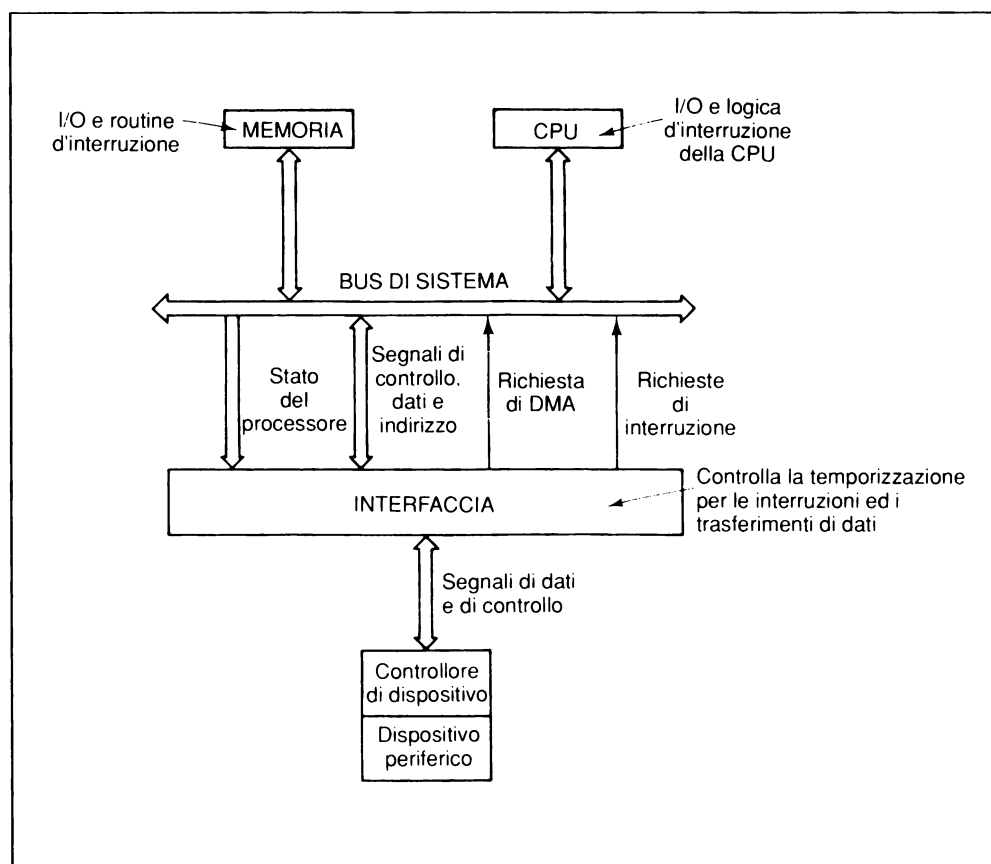



Fig. 13.1 *Organizzazione tipica di un'interfaccia.*

Le caratteristiche funzionali di un'interfaccia dipendono interamente dalla sua applicazione. Per operazioni standard come un trasferimento seriale di dati ai terminali CRT o il trasferimento parallelo ad unità a nastro e ad unità a disco, l'interfaccia ed eventualmente una parte del controllore del dispositivo sono solitamente racchiusi in un unico contenitore come chip periferico. Quando l'interfaccia serve a connettere dispositivi speciali, allora può rendersi necessaria la sua progettazione "ad hoc".

13.1.1 Progettazione funzionale di interfacce

La maggior parte dei dispositivi periferici standard sono connessi ad un sistema mediante chip periferici che realizzano le interfacce. Pertanto, è importante comprendere i princìpi operativi generali di tali chip, che funzionano come interfacce *programmabili.* Tali interfacce richiedono l'inizializzazione ed operano sotto il controllo della routine di I/O. La progettazione, l'integrazione ed il collaudo di queste interfacce e delle relative routine richiede la cooperazione tra il progettista dell'hardware ed il programmatore.

La Tab. 13.1 elenca alcuni elementi da specificare quando si descrive l'attività funzionale di un'interfaccia. Questa descrizione funzionale viene preparata solitamente dal progettista dell'hardware, che include i dettagli di programmazione richiesti per l'interfaccia.

La descrizione funzionale inizia con una discussione dello scopo dell'interfaccia. Questo dipende dai requisiti del progetto del sistema, in particolare dal tipo di

Tab. 13.1 *Progetto funzionale di un'interfaccia.*

Scopo dell'interfaccia

Modalità operative
- Inizializzazione
- Trasferimenti di dati ed altre operazioni

Temporizzazione

Caratteristiche elettriche

Caratteristiche fisiche

Programmazione: sequenza di dati e di comandi trasferiti

Procedura di test

dispositivo periferico interessato. Le modalità operative sono descritte definendo le operazioni della circuiteria necessarie per effettuare l'inizializzazione e i trasferimenti di dati. La descrizione non deve tralsciare le considerazioni di temporizzazione e simili aspetti dell'hardware.

Anche le caratteristiche elettriche e fisiche dell'interfaccia sono incluse nella descrizione funzionale. I dettagli elettrici sono determinati dal bus di sistema e dai requisiti elettrici del controllore del dispositivo. Tali dettagli includono la specificazione dei livelli di tensione, la rapidità di variazione dei segnali (tempi di salita), e simili fattori. Le considerazioni fisiche comprendono i requisiti ambientali e le limitazioni di spazio sulle piastre circuitali interessate. L'intervallo di temperatura e di umidità per l'ambiente determina il tipo di chip e d'incapsulamento richiesti per i circuiti integrati. Le dimensioni delle piastre circuitali influiscono sul numero di chip e sul loro posizionamento sulla piastra. La maggior parte dei progetti interessa piastre circuitali standard che s'inseriscono in connettori sul bus di sistema.

Una volta che la modalità operativa di un'interfaccia è stata specificata, possono essere definiti i requisiti di programmazione. Un'interfaccia è controllata da sequenze di variabili logiche o da valori binari scritti, cioè inviati, alla circuiteria d'interfacciamento da una routine di I/O. Dev'essere definita una sequenza per ogni modalità operativa. Per esempio, devono essere descritte la procedura di programmazione per il reset dell'interfaccia e l'inizializzazione a qualche stato noto. La programmazione di un trasferimento di dati richiede di determinare se l'interfaccia è pronta per il trasferimento, effettuare il trasferimento e verificare che non ci siano stati errori.

Infine si definisce una procedura di test per l'interfaccia. Di solito, essa include una procedura hardware per verificare il funzionamento dell'interfaccia, come pure i passi per eseguire il test dell'interfaccia sotto il controllo del programma. I chip periferici disponibili come parte della famiglia di prodotti della Motorola servono a semplificare la progettazione e la programmazione delle interfacce.

13.1.2 Chip periferici come interfacce

La famiglia dell'MC68020 comprende un certo numero di dispositivi periferici per l'interfacciamento. Questi circuiti integrati forniscono l'interfaccia a dispositivi particolari e fanno risparmiare una notevole quantità di progettazione dell'hardware relativo all'interfaccia. Questi chip sono progettati per essere connessi al bus di sistema dell'MC68020 direttamente o con l'aggiunta di una circuiteria minima.

La programmazione dei chip periferici avviene scrivendo o leggendo da registri interni ad essi. Gli indirizzi di tali registri sono contenuti nello spazio di I/O del sistema. Per sistemi basati sull'MC68020, questi indirizzi appaiono come locazioni di memoria, a causa del metodo di I/O rappresentato nella memoria adottato dall'MC68020.

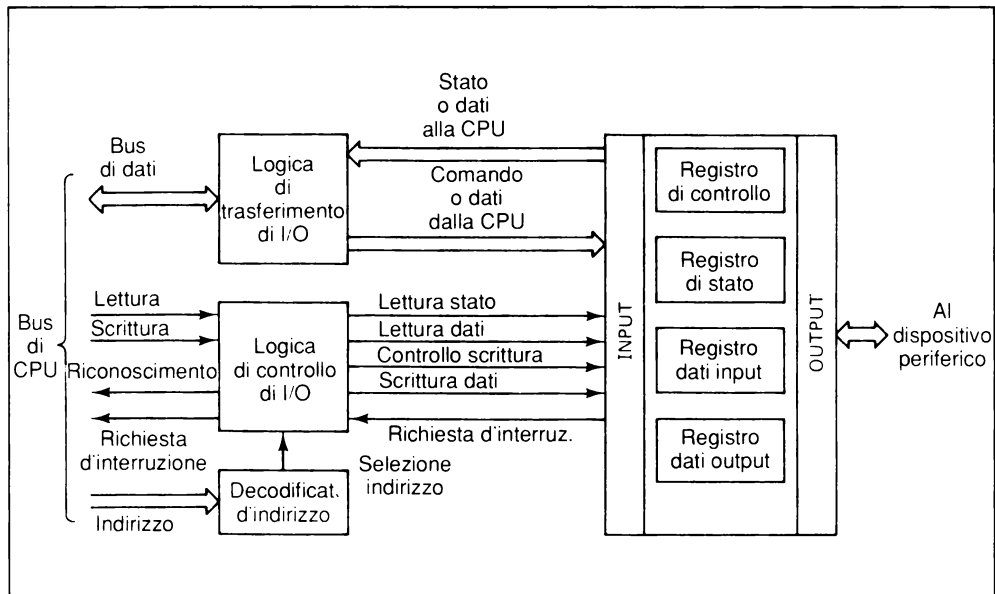



Fig. 13.2 *Un tipico chip periferico.*

La Fig. 13.2 illustra la struttura di un tipico chip periferico. Il lato di uscita (*output*) del chip periferico ha linee di controllo e di dati progettate per trasferire le informazioni di controllo e i dati ad un dispositivo periferico. Le linee di segnale sul lato d'ingresso (*input*) sono utilizzate per selezionare o "abilitare" il chip e per trasferire i dati tra la CPU e i registri interni del chip. Ogni registro interno ha un indirizzo di I/O definito durante il progetto del sistema. Il decodificatore d'indirizzo mostrato abilita il chip quando la CPU pone un indirizzo di registro interno sulle linee d'indirizzo.[1] Le linee di controllo di lettura/scrittura, originate dalla circuiteria logica di controllo di I/O, selezionano poi un particolare registro del chip. Allorché il chip periferico avrà riconosciuto la richiesta emessa dalla CPU, la logica di trasferimento di I/O agirà come un buffer per i dati tra il bus di dati ed il chip periferico.

Il registro di controllo riceve una sequenza di bit dalla CPU per controllare l'attività del chip. Per esempio, l'istruzione dell'MC68020

```
MOVE.B     #<d8>,REGC
```

potrebbe essere usata per trasferire 8 bit al registro di controllo situato all'indirizzo REGC. Vari bit nel byte di comando <d8> servono di solito a determinare se un'operazione di trasferimento è d'ingresso o di uscita e se le interruzioni richieste dal chip sono abilitate o no.

[1] In certi chip periferici, i registri di controllo e di stato hanno il medesimo indirizzo. Ciò si ottiene usando la linea di segnale di "lettura" per selezionare un registro di stato a sola lettura. Similmente, la linea di segnale di scrittura dovrebbe essere usata per selezionare il registro di controllo a sola scrittura. Molti chip periferici della famiglia dell'MC68020 sono dotati di questa caratteristica.

Un registro di stato sul chip contiene informazioni sul trasferimento o in merito al chip. I bit in tale registro potrebbero indicare se il chip è pronto per il trasferimento di dati, come pure lo stato d'interruzione e le condizioni di errore. Uno stato di bit potrebbe essere letto mediante l'istruzione

```
MOVE.B    REGS,D1
```

se REGS è l'indirizzo del registro di stato sul chip.

Due registri di dati sono mostrati sul chip nella Fig. 13.2. Uno di essi riceve i dati dal dispositivo periferico, mentre l'altro memorizza i dati da trasmettere al dispositivo. Il registro di dati d'ingresso viene letto in una maniera simile a quella impiegata per leggere il registro di stato. La CPU scrive nel registro di dati di uscita nello stesso modo in cui scrive nel registro di controllo.

13.1.3 I chip di supporto di sistema della Motorola

La Tab. 13.2 mostra un elenco parziale dei chip della famiglia di prodotti a 16 bit e a 32 bit della Motorola. Il diagramma a blocchi semplificato della Fig. 13.3 illustra le loro connessioni ai bus d'indirizzi e di dati dell'unità centrale di elaborazione (*Central Processing Unit:* CPU). Ciascuno di questi chip di supporto soddisfa una funzione specifica d'interfacciamento ed è programmato in conformità con i suoi requisiti peculiari. Naturalmente, un sistema tipico conterrà soltanto un insieme ristretto di chip, selezionato nella gamma di quelli disponibili.

Tab. 13.2 *La famiglia di chip della Motorola.*

Prodotto	N. Dispositivo	Denominazione
CPU	MC68000	Unità di microprocessore a 16 bit (CPU)
	MC68008	CPU a 16 bit con bus dati di 8 bit
	MC68010	CPU a 16 bit con memoria virtuale
	MC68020	CPU a 32 bit
	MC68030	CPU a 32 bit
MMU	MC68451	Unità di gestione della memoria (*Memory Management Unit:* MMU)
	MC68851	Unità di gestione della memoria impaginata
Processore matematico	MC68881	Coprocessore in virgola mobile (*Floating Point CoProcessor:* FPCP)
	MC68882	Coprocessore in virgola mobile

Tab. 13.2 *La famiglia di chip della Motorola. (continuazione)*

Prodotto	N. Dispositivo	Denominazione
Controllori di bus	MC68452	Modulo di arbitrato di bus (*Bus Arbitration Module:* BAM)
	MC68153	Modulo di generatore interruzione di bus (*Bus Interrupter Module:* BIM)
Controllori di DMA	MC68440	Controllore di DMA (a due canali) (DDMA)
	MC68450	Controllore di DMA (a quattro canali) (DMAC)
I/O di tipo generale	MC68230	Timer d'interfaccia parallela (*Parallel Interface Timer:* PI/T)
	MC68901	Periferica multifunzione (*MultiFunction Peripheral:* MFP)
Controllore di periferica	MC68120	Controllore di periferica intelligente (*Intelligent Peripheral Controller:* IPC)
Comunicazioni di dati	MC68561	Controllore di comunicazione multiprotocollo II (*Multiprotocol Communication Controller II:* MPCCII)
	MC68562	Controllori di comunicazioni seriali universali duali
	MC68564	Ingresso/Uscita seriale (*Serial Input/Output:* SIO)
	MC68652	Controllore di comunicazione multiprotocollo (*Multiprotocol Communication Controller:* MPCC)
	MC68653	Controllore di generatore di polinomi (*Polynomial Generator Checker:* PGC)
	MC68661	Interfaccia di comunicazioni programmabile potenziata (*Enhanced Programmable Communications Interface:* EPCI)
	MC68681	Ricetrasmettitore universale asincrono duale (*Dual Universal Asynchronous Receiver/Transmitter:* DUART)
Controllori di disco	MC68454	Controllore multidisco intelligente (*Intelligent Multiple-Disk Controller:* IMDC)
	MC68465	Controllore di disco flessibile (*Floppy Disk Controller:* FDC)

Nota: Non tutti i chip periferici prodotti dalla Motorola sono stati elencati.

Quando il sistema di computer comprende un'unità di gestione della memoria (*Memory Management Unit:* MMU), tale unità controlla l'emissione degli indirizzi sulle linee d'indirizzo. L'MC68030 combina le capacità della CPU e della gestione della memoria su un singolo chip. Questo processore sarà descritto nel cap. 15. Si rimanda il lettore alla letteratura tecnica della Motorola per i dettagli concernenti un particolare chip periferico tra quelli elencati nella Tab. 13.2.

FAMIGLIA DI M68000

ELABORAZ. MATEMATICA
MC68881 FPCP

ARBITRATO DI BUS
MC68452 BAM

MEMORIZZAZ. SU DISCO
MC68454 IMDC

CONTROLLORI
MC68120 IPC

CPU
MC68000
MC68008
MC68010
MC68020
MC68030

DATI

INDIRIZZO

MMU
GESTIONE DELLA MEMORIA

MC68440 DDMA
MC68450 DMAC
DMA

MC68230 PI/T
MC68901 MFP
M6800 PERIFERICHE
I/O DI TIPO GENERALE

MC68652 MPCC
MC68653 PGC
MC68661 EPCI
MC68681 DUART
MC68562 DUSCC
MC68564 SIO
COMUN. DATI

Note:
(1) La famiglia dell'M68000 denota processori sia da 16 che da 32 bit ed i chip periferici.
(2) L'MC68030 comprende sia il processore che l'unità di gestione della memoria su un singolo chip.

Fig. 13.3 *La famiglia di chip dell'MC68000.*

ESERCIZI

13.1.1

I sistemi basati sull'MC68020 impiegano l'I/O rappresentato nella memoria per il trasferimento di dati. Esiste un altro metodo, noto come *I/O isolato.* Con questa tecnica, l'insieme di istruzioni della CPU comprende delle istruzioni di I/O distinte. Solitamente ci sono varie istruzioni di I/O (IN e OUT) e 256 possibili locazioni di I/O, denominate *porte*. L'accesso a questa porte avviene mediante linee di segnale separate, che non fanno parte del bus di indirizzi del sistema.Si descrivano i vantaggi e gli svantaggi di ciascun metodo in termini della flessibilità del sistema e dei requisiti dell'hardware per l'interfacciamento.

13.1.2 Si descrivano i passi nel programma e la sequenza hardware necessaria ad inizializzare un'interfaccia per ricevere un byte di dati da un dispositivo esterno. Si definisca l'operazione dell'interfaccia e l'uso dei suoi registri dopo la ricezione del byte. Quale sequenza è richiesta dal processore stesso per la lettura del byte in questione?

13.1.3 Si consulti la letteratura tecnica della Motorola per descrivere gli scopi e le operazioni dei chip periferici disponibili per sistemi basati sull'MC68020.

13.2 PROGRAMMAZIONE DEI CHIP PERIFERICI

L'impiego dell'I/O rappresentato nella memoria in sistemi basati sull'MC68020 offre una grande flessibilità al programmatore di routine per il controllo di chip periferici. Qualsiasi istruzione che faccia riferimento alla memoria può essere usata per il controllo o per il trasferimento di dati da e verso un'interfaccia periferica. Il potente insieme di istruzioni e le numerose modalità d'indirizzamento dell'MC68020 possono essere applicate a tale programmazione. Inoltre, sono disponibili due istruzioni speciali per controllare ed accedere ai registri dei chip periferici: RESET e MOVEP (*MOVE Peripheral data:* trasferisci dati di periferica), che sono presentate all'inizio di questo paragrafo.

Una discussione generale dei trasferimenti di I/O seguirà nel par. 13.2.2. La distinzione fra trasferimenti avviati dalla CPU e trasferimenti avviati dal dispositivo sarà presentata in quel paragrafo. Le tecniche di I/O trattate sono di natura generale poiché non dipendono dalle caratteristiche specifiche di un chip periferico.

Il modulo MVME133 è descritto nel par. 13.2.3 per illustrare la capacità di I/O ed altre caratteristiche di un tipico computer su piastra singola. Come esempio di un chip periferico speciale, sarà considerata in un certo dettaglio la periferica multifunzione MC68901. Il suo impiego come chip periferico sarà dimostrato da un programma di esempio.

13.2.1 Le istruzioni RESET e MOVEP

Le due istruzioni elencate nella Tab. 13.3 sono impiegate per l'accesso o il controllo dei dispositivi periferici. L'istruzione RESET attiva una linea di segnale della CPU, utilizzata per far sì che le interfacce assumano il loro stato hardware iniziale. Questa istruzione può essere eseguita soltanto nella modalità di supervisore e non interessa lo stato del processore. L'esecuzione prosegue con l'istruzione successiva.

Tab. 13.3 *Le istruzioni RESET e MOVEP.*

Sintassi	Operazione
RESET	Attivazione della linea di segnale RESET
MOVEP.<l> <Dn>,<d_{16}>(An)	Trasferimento a (EA), (EA + 2), ...
MOVEP.<l> <d_{16}>(An),<Dn>	Trasferimento da (EA), (EA + 2), ...

Note:
1. RESET è un'struzione privilegiata.
2. <l> è W o L per MOVEP.

L'istruzione MOVEP (*MOVE Peripheral data:* trasferisci dati di periferica) trasferisce un valore di word o di longword tra un registro di dati e byte *alternati* di memoria. Questa istruzione è usata per accedere a chip periferici i cui registri hanno indirizzi di byte successivi pari o dispari nella memoria. Per esempio, l'istruzione

```
MOVEP.L        D1,0(A1)
```

trasferisce quattro byte da D1 ad ogni byte alternato, a partire dal primo byte indirizzato da (A1). Il byte superiore alto, (D1)[31:24], è trasferito alla locazione (A1); il byte superiore intermedio, (D1)[23:16], all'indirizzo di byte (A1) + 2; e così via. Questo metodo di accesso ai registri d'interfaccia è impiegato per la famiglia di chip periferici dell'M6800 a 8 bit e per alcuni chip periferici a 16 bit prodotti dalla Motorola.

13.2.2 Tecniche di trasferimento di I/O

I trasferimenti di I/O sono suddivisi in quelli avviati dalla CPU e in quelli avviati dal dispositivo periferico e dal suo controllore. La Tab. 13.4 elenca le tecniche di trasferimento in queste categorie e definisce l'inizializzazione richiesta e le operazioni del programma. Prima che inizi il trasferimento, la routine di I/O eseguita dalla CPU svolge l'inizializzazione dell'interfaccia.

Il trasferimento *incondizionato* richiede che il dispositivo periferico sia pronto per i trasferimenti in qualsiasi istante. Un esempio comune di questo tipo di trasferimento si trova nei sistemi in cui il "dispositivo" è un'unità per visualizzare numeri o caratteri. La routine di I/O si limita a trasferire i dati con un'istruzione MOVE alla routine appropriata. La circuiteria dell'unità di visualizzazione è impiegata per con-

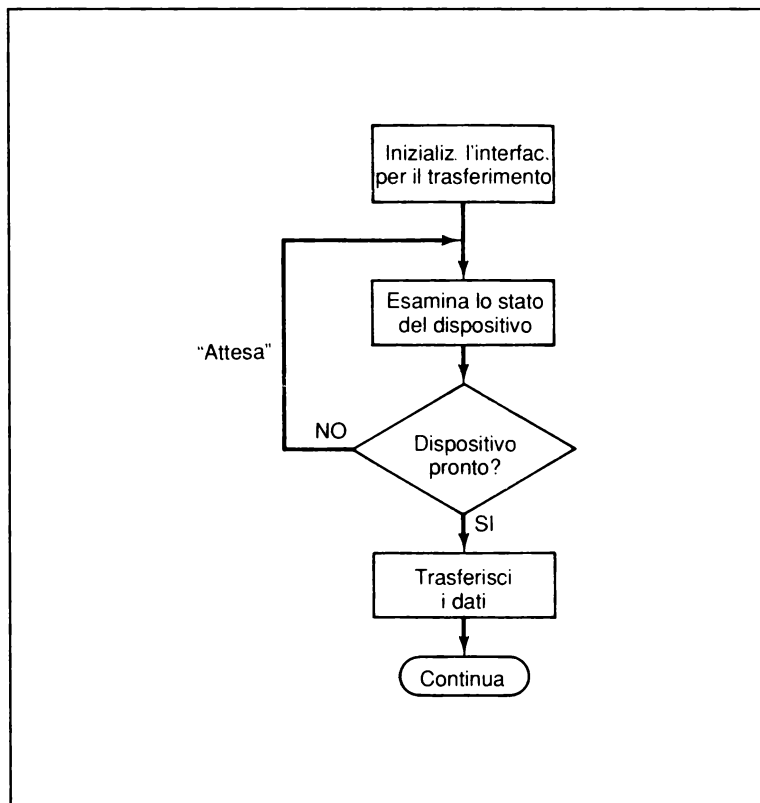
Tab. 13.4 *Tecniche di trasferimento di I/O.*

TIPO DI TRASFERIMENTO	INIZIALIZZAZIONE DAL PROGRAMMA	OPERAZIONE DEL PROGRAMMA
Trasferimento avviato dalla CPU		
Incondizionato	Nessuna	Trasferimento di dati.
Condizionato	Inizializza il dispositivo per la direzione di trasferimento	Test dello stato del dispositivo e attesa finché è pronto; poi trasferimento.
Trasferimento avviato dal dispositivo		
Trasferimento d'interruzione	1. Inizializza il dispositivo per il trasferimento di I/O con interruzione 2. Abilita le interruzioni	1. Trasferimento di dati quando avviene l'interruzione. 2. Cancella la richiesta d'interruzione dopo aver eseguito il trasferimento.
DMA	1. Inizializza il dispositivo per il trasferimento di I/O 2. Carica i registri di DMA (a) Conto (b) Indirizzo 3. Invia il comando d'inizio	Elabora l'interruzione di fine del blocco.

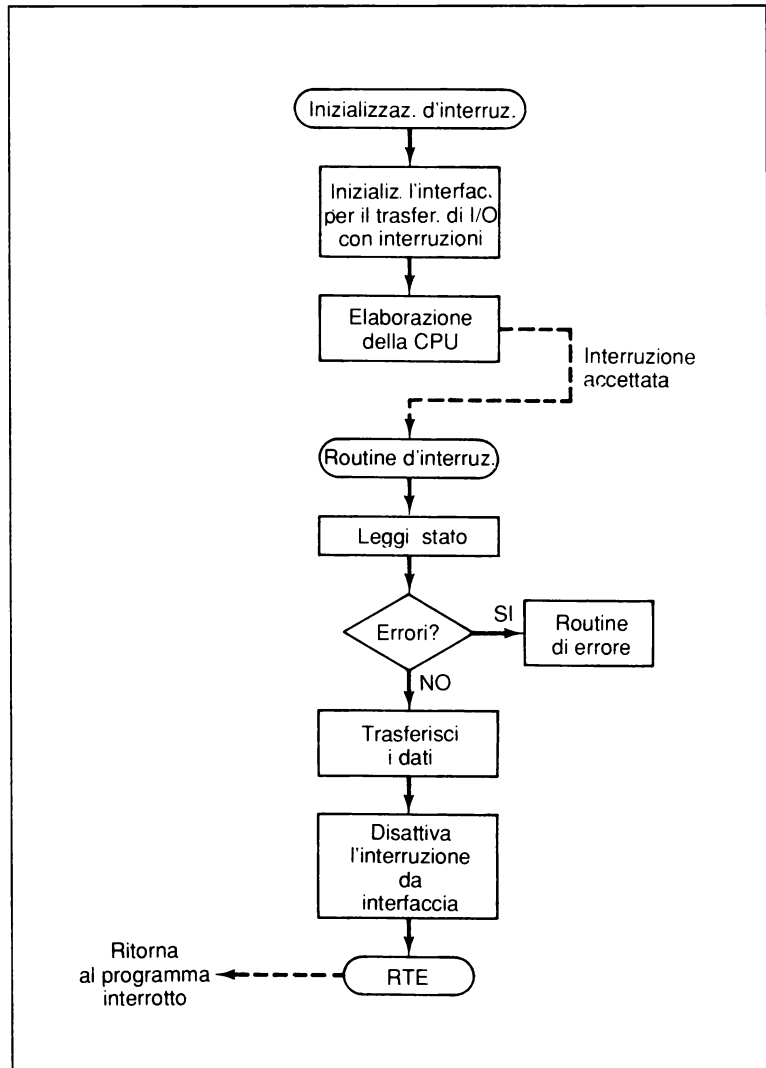
Nota: L'I/O condizionato è talvolta definito I/O programmato o I/O a interrogazione ciclica.

vertire nel formato di visualizzazione la word binaria proveniente dalla CPU. Un altro impiego di questo metodo è la lettura di un gruppo di interruttori le cui posizioni (aperto/chiuso) sono codificate in una sequenza binaria.

I trasferimenti condizionati sono talvolta denotati come *I/O programmato* o anche *I/O a interrogazione ciclica* (*polled I/O*). Le operazioni di questi trasferimenti sono illustrate nella Fig. 13.4. Una volta che l'interfaccia è stata inizializzata, la routine di I/O verifica ripetutamente il registro di stato del chip finché lo stato non indica che il dispositivo è pronto. La routine trasferisce quindi il dato al dispositivo periferico. Poiché la CPU è in un ciclo di attesa finché il dispositivo non è pronto, l'utilità di questo trasferimento è limitata. Per esempio, un trasferimento condizionato potrebbe essere utilizzato per scrivere in un registro di controllo durante l'inizializzazione dell'interfaccia.

Fig. 13.4
Trasferimento condizionato.

Per dispositivi che trasmettono o ricevono dati molto lentamente rispetto ai tempi di esecuzione delle routine di I/O, si preferisce il trasferimento controllato da interruzione. Questo metodo di trasferimento è mostrato nella Fig. 13.5. L'interfaccia viene dapprima inizializzata per trasferire i dati; dopodiché, la CPU esegue altri programmi finché non si presenta un'interruzione. Quando il controllo è passato alla routine d'interruzione, viene effettuato un test sullo stato del chip periferico per scoprire eventuali errori. Qualora sia rivelato un errore, allora viene intrapresa l'azione appropriata; altrimenti, ha luogo il trasferimento. In seguito, la richiesta d'interruzione dell'interfaccia dev'essere rimossa, prima che possa aver luogo il trasferimento successivo. Ciò di solito avviene automaticamente all'atto della lettura del registro di stato del chip periferico.

Quando un dispositivo è in grado di trasferire blocchi di dati ad alta velocità, s'impiega spesso il metodo di trasferimento di accesso diretto alla memoria (*Direct Memory Access:* DMA). Un'interfaccia di DMA contiene solitamente due registri programmabili: un registro di contatore, col numero dei valori da trasferire, ed un registro d'indirizzo, in cui è memorizzato l'indirizzo iniziale del blocco. Il programma d'inizializzazione carica questi registri ed invia una sequenza di comandi per

Fig. 13.5 *Trasferimento controllato da interruzione.*

avviare il trasferimento. La CPU sarà quindi libera di elaborare altri programmi. Il trasferimento di dati tra l'interfaccia e la memoria è totalmente controllato dalla circuiteria di DMA, senza interventi da parte della CPU.

Per ogni trasferimento in DMA di un valore di dati, la circuiteria di DMA richiede l'uso del bus di sistema. La CPU rilascia il bus per la durata del trasferimento, che solitamente è di un ciclo di bus. Durante il trasferimento, l'interfaccia di DMA controlla la memoria così come fa normalmente la CPU. Tipicamente, meno di un ciclo di bus su cinque è impiegato per i trasferimenti di DMA. Questo "furto di ciclo" ha un effetto irrilevante sulle prestazioni del sistema, nella maggior parte dei casi.

Per un confronto, l'MC68020 richiede almeno tre cicli di bus per prelevare un'istruzione. Al termine del trasferimento di un intero blocco di dati, la circuiteria di DMA causa un'interruzione. La CPU potrà allora elaborare il dato d'ingresso o avviare il successivo trasferimento di DMA.

13.2.3 Caratteristiche del modulo MVME133

Il computer su piastra singola MVME133 è stato presentato nel par. 1.1 come esempio di un tipico modulo di computer. Nella presente discussione, saranno considerate le caratteristiche di tale modulo e la programmazione dei chip periferici. I chip più importanti e la circuiteria per l'MVME133 sono connessi come mostrato nella forma di diagramma a blocchi nella Fig. 13.6. Gli elementi connessi al bus interno del modulo sono il chip MC68901 (timer, porta di debug, e controllo/stato), la circuiteria d'interruzione, il monitor 133BUG (ROM), l'MC68881, i dispositivi di I/O seriale, 1 megabyte di RAM, e l'unità di microprocessore (MPU) MC68020. Gli altri componenti sulla piastra servono come interfacce tra il bus interno ed i chip ed il VMEbus (connettori P1 e P2). Questa porzione del modulo MVME133 sarà descritta nel cap. 14.

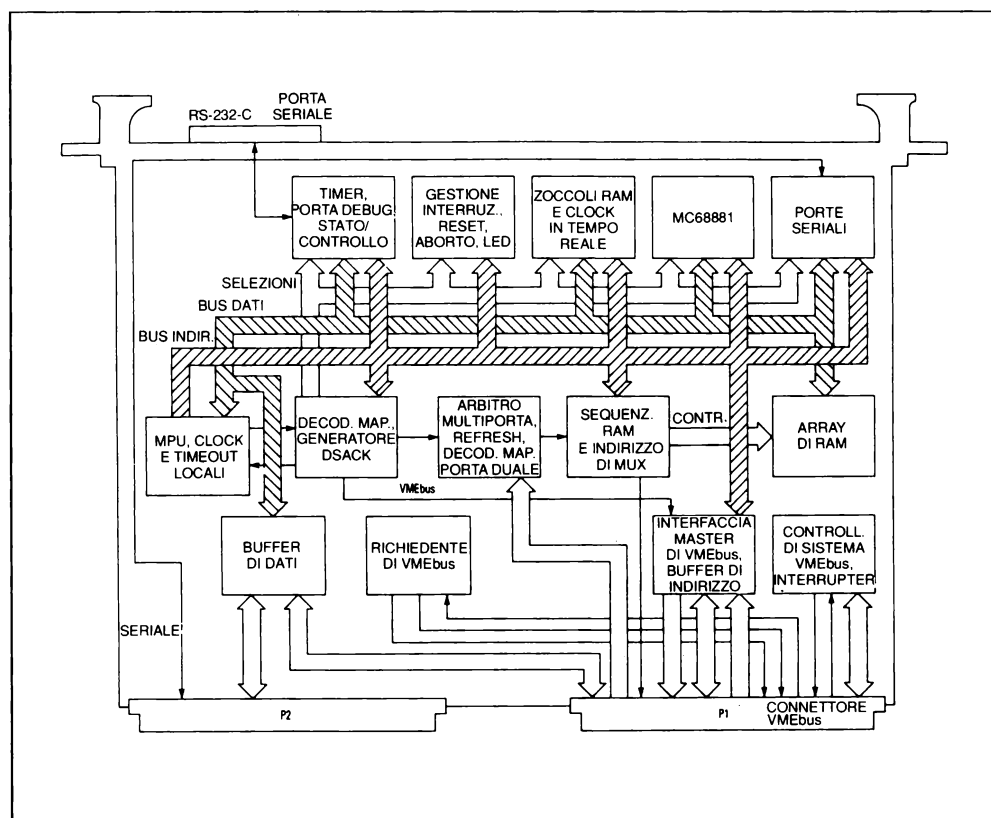



Fig. 13.6 *Diagramma a blocchi del modulo MVME133. (Per gentile concessione di Motorola, Inc.)*

Il connettore della porta seriale (RS232) è quello tra l'MVME133 ed il terminale dell'operatore, descritto nel par. 7.6. Per effettuare i trasferimenti d'ingresso/uscita di caratteri, l'MC68901 dev'essere controllato da una routine di I/O che programmi la porzione seriale del chip per l'ingresso o per l'uscita, come richiesto dalla direzione del trasferimento. Nel par. 7.6, le routine di I/O erano parte del monitor BUG133. Quindi, un programmatore che desideri eseguire i trasferimenti di I/O non ha fatto altro che richiedere il trasferimento inserendo un'istruzione della forma TRAP #15 in un programma. L'istruzione TRAP ha causato l'esecuzione dell'appropriata routine di I/O del monitor 133BUG. Il presente paragrafo tratta la progettazione delle routine di I/O per effettuare i trasferimenti.

La Fig. 13.7 mostra il diagramma a blocchi dell'MC68901. Questo chip è denominato "periferica multifunzione" poiché esegue sia il trasferimento di I/O che le funzioni di temporizzazione sotto il controllo del programma.

Nota: L'USART (ricetrasmettitore sincrono/asincrono universale) è impiegato per i trasferimenti seriali di I/O.

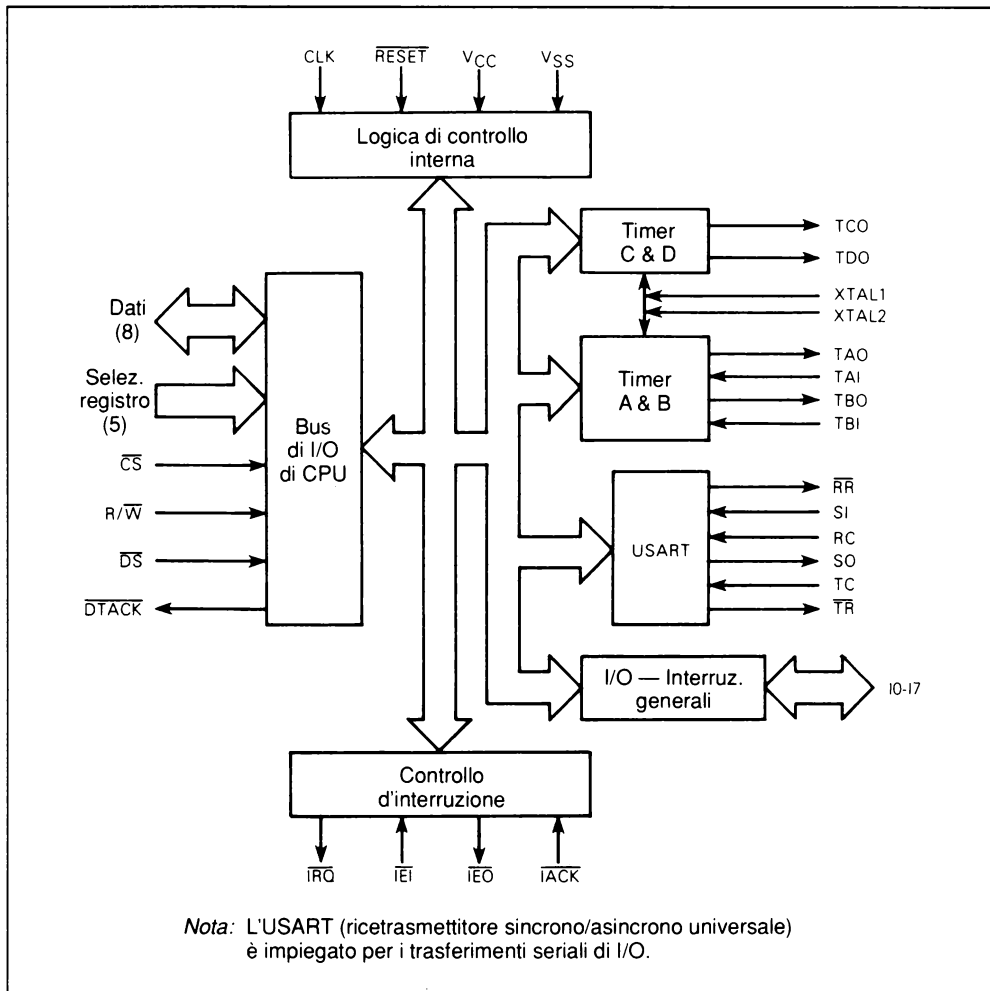



Fig. 13.7 *Diagramma a blocchi dell'MC68901. (Per gentile concessione di Motorola, Inc.)*

Per indirizzare uno dei suoi registri, un programma seleziona un indirizzo specifico entro lo spazio d'indirizzi del modulo MVME133. L'esempio 13.1 include un programma per l'MC68901.

La mappa d'indirizzi dell'MVME133. La Tab. 13.5 mostra gli indirizzi associati col modulo MVME133. Il modulo ha 24 linee di segnali d'indirizzo, che consentono un intervallo d'indirizzamento da $00 0000 a $FF FFFF. L'MC68901 è indirizzato tra $F8 0000 e $F8 002F ad indirizzi di byte dispari.

La mappa d'indirizzi dell'MC68901. I registri dell'MC68901 sono indirizzati come mostrato nella Tab. 13.6. Per indirizzare un registro specifico, l'indirizzo di base ($F80000) viene aggiunto all'offset esadecimale indicato nella tabella. Per esempio, l'istruzione

```
MOVE.B    D1,$F8001D
```

copia il valore di un byte contenuto in D1 nel registro di controllo per i timer C e D.

I/O seriale con l'MC68901. La periferica multifunzione MC68901 opera come un chip di I/O seriale, come mostrato nella Fig. 13.8. In questa configurazione, il chip trasmette o riceve i dati seriali quando è connesso ed un'unità periferica esterna come un terminale di operatore. Internamente, il chip riceve i bit seriali come ingressi e li converte nella forma parallela di 8 bit. I bit costituiscono un byte di dati che sarà letto dalla CPU tramite il bus di sistema. Per l'uscita, il chip converte in forma seriale il byte di dati di 8 bit provenienti dalla CPU, per la trasmissione all'unità periferica.

Tab. 13.5 *Assegnazione della memoria dell'MVME133.*

Elemento	Indirizzo
Area di RAM	$000000-$0FFFFF[1]
VMEbus	$100000-$EFFFFF
Area di ROM	$F00000-$F3FFFF $F40000-$F7FFFF[2]
Periferica multifunzione MC68901	$F80000-$F8002F $F80030-$F9FFFF[2]
Vari	$FA0000-$FFFFFF

Note:
1. Gli indirizzi da $000000 a $000007 servono a contenere i vettori di reset.
2. Questi indirizzi non sono normalmente utilizzati.

Tab. 13.6 *Mappa di registri dell'MC68901.*

Indirizzo							
Hex	Binario						
	RS5	*RS4*	*RS3*	*RS2*	*RS1*	*Abbreviazione*	*Nome del registro*
01	0	0	0	0	0	GPIP	I/O generale
03	0	0	0	0	1	AER	Fronte attivo
05	0	0	0	1	0	DDR	Direzione del dato
07	0	0	0	1	1	IERA	Abilitazione d'interruzione A
09	0	0	1	0	0	IERB	Abilitazione d'interruzione B
0B	0	0	1	0	1	IPRA	Interruzione in sospeso A
0D	0	0	1	1	0	IPRB	Interruzione in sospeso B
0F	0	0	1	1	1	ISRA	Interruzione in servizio A
11	0	1	0	0	0	ISRB	Interruzione in servizio B
13	0	1	0	0	1	IMRA	Maschera d'interruzione A
15	0	1	0	1	0	IMRB	Maschera d'interruzione B
17	0	1	0	1	1	VR	Vettore
19	0	1	1	0	0	TACR	controllo di timer A
1B	0	1	1	0	1	TBCR	controllo di timer B
1D	0	1	1	1	0	TCDCR	controllo di timer C e D
1F	0	1	1	1	1	TADR	Dati di timer A
21	1	0	0	0	0	TBDR	Dati di timer B
23	1	0	0	0	1	TCDR	Dati di timer C
25	1	0	0	1	0	TDDR	Dati di timer D
27	1	0	0	1	1	SCR	Cararatteri sincroni
29	1	0	1	0	0	UCR	Controllo di USART
2B	1	0	1	0	1	RSR	Stato del ricevitore
2D	1	0	1	1	0	TSR	Stato del trasmettitore
2F	1	0	1	1	1	UDR	Dati di USART

Nota: RS1-RS5 sono linee di segnale connesse alle linee di segnali d'indirizzo A1-A5 della CPU.

Fonte: Motorola, Inc.

La Fig. 13.8 mostra i registri dell'MC68901 che partecipano al trasferimento di I/O seriale. Il chip viene inizializzato programmando le caratteristiche di temporizzazione mediante il timer C ed il protocollo per i dati trasmessi o ricevuti, usando il registro di controllo dell'USART (*USART Control Register:* UCR). I registri di stato per i dati ricevuti (*Received Status Register:* RDR) e per i dati trasmessi (*Transmitted Status Register:* TSR) sono usati per determinare l'istante in cui il chip ha ricevuto un byte o è pronto a trasmettere il byte successivo, rispettivamente.

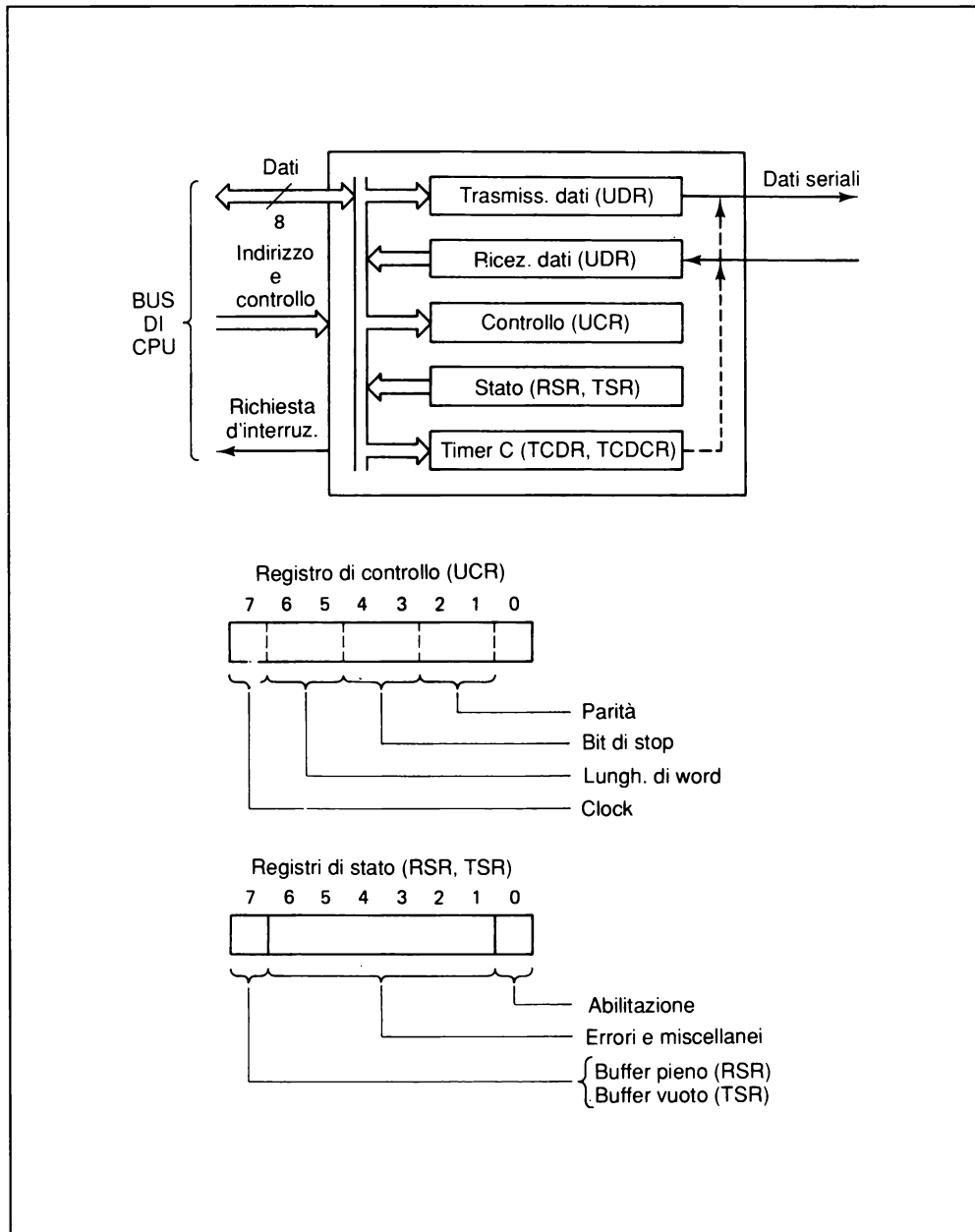



Fig. 13.8 *Diagramma semplificato dell'MC68901 come un chip di I/O seriale.*

Per spiegazioni più dettagliate, si rimanda il lettore alla pubblicazione *MC68901 MFP User's Manual* della Motorola, Inc.

Esempio 13-1

Le subroutine in Fig. 13.9 illustrano le tecniche fondamentali per la programmazione dell'ingresso/uscita della periferica multifunzione (*Multi-Functional Peripheral:* MFP) MC68901. È incluso un programma di test che chiama varie subroutine per inizializzare l'MC68901, acquisire un carattere dal terminale dell'operatore, elaborare il carattere, e infine echeggiare il carattere sul terminale. L'indirizzo di base dell'MC68901 nello spazio d'indirizzi della piastra di CPU MVME133 è $F80000. I vari registri dell'MC68901 sono indirizzati come offset da questo indirizzo di base, come mostrato nella Tab. 13.6.

L'MC68901 è programmato per la velocità di trasferimento (*baud rate*) ed altre caratteristiche della comunicazione seriale, scrivendo le configurazioni di bit appropriate nei suoi registri di controllo. La subroutine INIT svolge tale funzione ed avvia i clock di baud rate per il ricevitore ed il trasmettitore dell'USART. Il baud rate è determinato da un segnale di clock controllato da un cristallo di quarzo; questo segnale viene immesso nell'MC68901, opportunamente ridotto in scala per fornire la frequenza appropriata degli impulsi di temporizzazione. Nel caso della pistra MVME133, la frequenza di clock per l'MC68901 è di 1.23 MHz. Una velocità di trasmissione di 9600 baud richiede una frequenza di:

$$\frac{1.23 \text{ MHz}}{9600 \text{ Hz}} = 128$$

come fattore di divisione, poiché la velocità del segnale di 9600 bit/secondo coincide col baud rate per l'I/O seriale in questo caso. Tale operazione sarà eseguita dividendo per 8 la frequenza di clock controllata dal cristallo, usando il timer C dell'MC68901 e poi dividendo per 16 l'uscita del timer per controllare i clock dell'USART. Le configurazioni di bit necessarie a tal fine sono determinabili dall'*User's Manual* dell'MC68901.

La subroutine INIT serve per programmare il timer C dell'MC68901 e per definire il protocollo di comunicazione. Il codice scritto nel registro di controllo dell'USART (*USART Control Register:* UCR) definisce i seguenti parametri:

a) Divisione per 16 (divisiore di clock)
b) Comunicazione asincrona
c) Caratteri di 8 bit
d) Nessuna verifica di parità
e) Numero di bit di start e di stop

Successivamente, i clock del ricevitore e del trasmettitore vengono avviati scrivendo i valori appropriati nei registri di stato. L'MC68901 è ora pronto per ricevere o trasmettere un carattere.

```
                          1.         TTL     FIGURA 13.9
                          2.         LLEN    100
                          3.  *
                          4.  *   ROUTINE DI I/O DI MFP DELL'MC68901
                          5.  *       - RICEVE IL CARATTERE DAL TERMINALE (INCHR)
                          6.  *       - INVIA IL CARATTERE IN USCITA (OUTCHR)
                          7.  *       - IL CARATTERE VIENE RICEVUTO ED INVIATO DA (D0.B)
                          8.  *
                          9.  *   PROGRAMMA DI TEST
                         10.  *
00010000                 11.         ORG     $10000
00010000  4EB9 00008000  12.         JSR     INIT            ;INIZIALIZZA MC68901
00010006  4EB9 0000802A  13.         JSR     INCHR           ;INPUT DI 1 CARATTERE
0001000C  4EB9 0000804E  14.         JSR     PROCESS         ;ELABORA L'INPUT
00010012  4EB9 0000803C  15.         JSR     OUTCHR          ;OUTPUT DI 1 CARATTERE
00010018  4E4F           16.         TRAP    #15
0001001A  0063           17.         DC.W    $0063           ;RITORNA AL MONITOR
                         18.  *
                         19.  *      INIZIALIZZA L'MC68901
                         20.  *
00008000                 21.         ORG     $8000
    = 00F80000           22.  MFPBASE EQU    $F80000         ;INDIRIZZO BASE DI MFP
    = 00F8001D           23.  TCDCR  EQU     MFPBASE+$1D     ;CONTROLLO DI TIMER C/D
    = 00F80023           24.  TCDR   EQU     MFPBASE+$23     ;DATI DI TIMER
    = 00F80029           25.  UCR    EQU     MFPBASE+$29     ;CONTROLLO DI USART
    = 00F8002B           26.  RSR    EQU     MFPBASE+$2B     ;STATO DI RICEVITORE
    = 00F8002D           27.  TSR    EQU     MFPBASE+$2D     ;STATO DI TRASMETTITORE
    = 00F8002F           28.  UDR    EQU     MFPBASE+$2F     ;DATI DI USART
                         29.  *
                         30.  *   INIZIALIZZA
                         31.  *       9600 BAUD, 8 BIT, NESSUNA PARITA'
                         32.  *       AVVIA I CLOCK
                         33.  *
00008000  13FC 0001      34.  INIT   MOVE.B  #$01,TCDR       ;CONTO=1
          00F80023
00008008  13FC 0011      35.         MOVE.B  #$11,TCDCR      ;DIVIDE PER 4
          00F8001D
00008010  13FC 0088      36.         MOVE.B  #$88,UCR        ;/16, ASINCRONO, 8 BIT
          00F80029
                         37.  *                              ;NESSUNA PARITA', 1 BIT DI STOP
00008018  13FC 0001      38.         MOVE.B  #$01,RSR        ;AVVIA IL CLOCK DEL RICEVITORE
          00F8002B
00008020  13FC 0005      39.         MOVE.B  #$05,TSR        ;AVVIA IL CLOCK DEL TRASMETTITORE
          00F8002D
00008028  4E75           40.         RTS

                         41.  *
                         42.  *   ATTENDE IL CARATTERE IN INGRESSO E LO TRASFERISCE IN D0
                         43.  *
0000802A  0839 0007      44.  INCHR  BTST.B  #7,RSR          ;BUFFER PIENO?
          00F8002B
00008032  67 F6          45.         BEQ.S   INCHR           ;NO: RESTA IN ATTESA
00008034  1039 00F8002F  46.         MOVE.B  UDR,D0          ;SI: MEMORIZZA IL CARATTERE
0000803A  4E75           47.         RTS
                         48.  *
                         49.  *   INVIA IL CARATTERE IN USCITA DA D0
                         50.  *
0000803C  0839 0007      51.  OUTCHR BTST.B  #7,TSR          ;BUFFER VUOTO?
          00F8002D
00008044  67 F6          52.         BEQ.S   OUTCHR          ;NO: RESTA IN ATTESA
00008046  13C0 00F8002F  53.         MOVE.B  D0,UDR          ;SI: LO INVIA
0000804C  4E75           54.         RTS
                         55.  *
                         56.  *   ELABORA IL CARATTERE DI INGRESSO
                         57.  *
0000804E  4E71           58.  PROCESS NOP                    ;ROUTINE FITTIZIA
                         59.  *
                         60.  *   (QUI VERIFICA PER CARATTERI SPECIALI, ECC.)
                         61.  *
00008050  4E75           62.         RTS
00008052                 63.         END
```

Fig. 13.9 *Programma di I/O per l'MC68901.*

Il programma di test chiama la subroutine INCHR per determinare se l'operatore ha battuto il tasto di un carattere. Se il buffer dell'USART è pieno, il carattere viene trasferito al registro D0. Altrimenti, la routine continua a esaminare il registro di stato del ricevitore (*Receiver Status Register:* RSR) finché non viene ricevuto un carattere. Dopo che un carattere è stato ricevuto, nell'esempio viene chiamata la subroutine PROCESS per eseguire qualsiasi elaborazione necessaria. Normalmente, l'elaborazione dovrebbe includere la verifica del carattere per determinare se sia stato ricevuto qualche carattere speciale. Per esempio, un carattere BREAK potrebbe servire ad indicare che l'operatore desidera terminare la sequenza d'inserimento e tornare al programma monitor.

Il carattere viene visualizzato sul terminale video dell'operatore allorché viene chiamata la subroutine OUTCHR. Qualsiasi carattere contenuto nel byte meno significativo di D0 viene trasmesso al terminale quando il buffer del trasmettitore è vuoto.

ESERCIZI

13.2.1 Si confrontino l'interruzione, il DMA ed il trasferimento condizionato per quanto concerne:

(a) La velocità operativa del sistema
(b) La complessità della programmazione
(c) La complessità dell'interfaccia

13.2.2 Si supponga che l'elaborazione da parte della CPU richieda 10 microsecondi per ogni byte di dati trasferiti. Quale sarebbe il metodo migliore per trasferire i dati per i seguenti dispositivi, con le velocità di trasferimento espresse tra parentesi in byte/secondo?

(a) Collegamento di comunicazione (1000)
(b) Unità a nastro (10000)
(c) Unità a disco (1000000)

13.2.3 Si tracci il grafo di flusso per una routine di trasferimento di I/O condizionato quando dev'essere trasferita una stringa di caratteri. Se un carattere è pronto ogni 0.1 secondi, si calcoli il tempo richiesto per inserire 10 caratteri, sapendo che il tempo di elaborazione della CPU è di 10^{-5} secondi per ogni carattere. Questa temporizzazione è tipica di un terminale lento.

13.2.4 Si utilizzi l'istruzione MOVEP dell'MC68020 per trasferire una tabella di valori da un'area della memoria ad un'altra area della memoria o a un dispositivo periferico. La tabella di valori ha *M* byte situati a indirizzi pari consecutivi che devono essere depositati ad indirizzi dispari consecutivi. Si trasferiscano quattro byte alla volta e si assuma che *M* sia divisibile per 4.

13.2.5

I dettagli di programmazione per un particolare chip periferico sono reperibili nello *User's Manual* (manuale per l'utente) del chip. Il lettore scelga uno o più dei possibili programmi elencati di seguito e li provi utilizzando il sistema basato sull'MC68020 di cui dispone.

(a) Creare di una routine che acquisisca dei caratteri in ingresso e li emetta in uscita usando un'istruzione TRAP #5.
(b) Modificare la routine della parte (a) per utilizzare l'elaborazione d'interruzione per l'ingresso e l'uscita.
(c) Creare un clock per visualizzare l'ora del giorno sul terminale dell'operatore.

13.3 TEMPI DI ESECUZIONE DELLE ISTRUZIONI

La temporizzazione della CPU MC68020 è controllata da un clock principale. Questo clock è un circuito che genera una sequenza periodica di impulsi che sincronizzano tutte le variazioni nelle linee di segnale del processore. La durata temporale di ogni operazione della CPU è determinata dal numero di impulsi di clock richiesti per l'operazione. Se il numero di impulsi al secondo (noto come frequenza degli impulsi) viene aumentato, la velocità operativa del processore aumenta. Similmente, se la frequenza viene diminuita, la CPU opera più lentamente.

La Fig. 13.10 illustra il segnale di clock per un caso tipico. La tabella inclusa nella figura elenca la frequenza operativa in megahertz (MHz) e la durata del ciclo (*cycle time*) o periodo del clock del processore per versioni selezionate della CPU. La frequenza operativa è talvolta denominata *velocità operativa* del processore.[2] Un MC68020 designato come MC68020RC16 opera alla massima frequenza di 16.7 milioni di impulsi di clock al secondo. La durata del ciclo è il reciproco della frequenza e rappresenta il *periodo* di un impulso di clock. Come mostrato nella Fig. 13.10, il tempo del ciclo per la versione a 16.7 MHz dell'MC68020 è indicato come 60 ns (nanosecondi).[3] Pertanto, la massima frequenza operativa corrisponde al minimo periodo di ciclo, come previsto. Un chip di circuito integrato come l'MC68020 ha anche una frequenza operativa minima, indicata come 8 MHz per l'MC68020RC16. Per questo valore minimo, la durata del ciclo aumenta a 125 ns.

Temporizzazione delle istruzioni. Il tempo richiesto da un'istruzione, T_{cyc}, può essere calcolato in termini della durata del ciclo t_{cyc} come:

$$T_{istr} = (\text{numero di cicli di clock}) \times t_{cyc}$$

[2] La frequenza operativa in hertz è considerata il reciproco della durata del ciclo. Più precisamente, essa denota la frequenza dell'onda sinusoidale fondamentale in un'analisi della serie di Fourier della forma d'onda.

[3] Una frequenza operativa più esatta è 16.67 MHz. Questo valore è impiegato nei calcoli di temporizzazione.

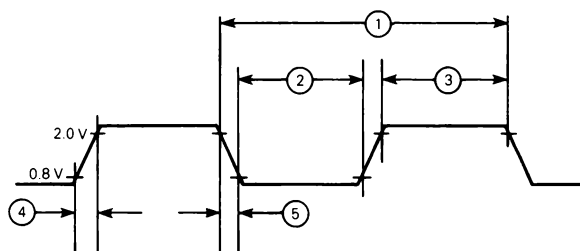
Num.	Caratteristiche	Simbolo	MC68020RC12		MC68020RC16		Unità
			Min	Max	Min	Max	
	Frequenza operativa	f	8	12.5	8	16.7	MHz
1	Periodo di clock	t_{cyc}	80	125	60	125	ns
2, 3	Durata dell'impulso di clock	t_{CL}, t_{CH}	32	125	24	95	ns
4, 5	Tempi di salita e di discesa	t_{Cr}, t_{Cf}	–	5	–	5	ns

Nota:
Le misure di temporizzazione sono effettuate con riferimento ad una tensione bassa di 0.8 V e ad una tensione alta di 2.0 volt, a meno che non sia specificato altrimenti. L'escursione di tensione in questo intervallo dovrebbe partire dall'esterno ed attraversare l'intervallo in modo che la salita o la discesa sia lineare tra 0.8 V e 2.0 V.

Fig. 13.10 *Forma d'onda di clock. (Per gentile concessione di Motorola, Inc.)*

dove i tempi sono espressi in secondi. Per ciascuna istruzione, il numero di cicli di clock richiesti è pubblicato nell'*MC68020 User's Manual.* Tuttavia, a causa della complessità dell'MC68020, la somma dei tempi delle istruzioni in base ai numeri pubblicati di cicli di clock fornisce raramente un risultato preciso quando si tenta di stabilire il tempo di esecuzione di una serie di istruzioni in un programma.

Come descritto nel par. 4.1, il pipeline dell'MC68020 e la memoria cache sono progettati per accelerare l'esecuzione di una sequenza di istruzioni. I fattori principali che influiscono sulla temporizzazione delle istruzioni dell'MC68020 sono i seguenti:

(a) Memoria cache (se abilitata)

(b) Prelievo anticipato (*prefetch*) delle istruzioni

(c) Sovrapposizione dell'esecuzione delle istruzioni

Questi fattori dipendono dal progetto della CPU e normalmente non sono sotto il controllo di un programma in esecuzione. Altri fattori, come la durata del ciclo di memoria, l'allineamento degli operandi, e le modalità d'indirizzamento e le dimensioni degli operandi selezionate da un programma, influiscono anch'essi sulla temporizzazione delle istruzioni. Tali fattori secondari non sono considerati qui poiché dipendono dalle caratteristiche di un sistema particolare e dal programma in esecuzione.

Il numero pubblicato di cicli di clock per l'MC68020 è suddiviso in tre categorie nell'*MC68020 User's Manual*. Le categorie per la temporizzazione delle istruzioni sono le seguenti:

(a) Caso migliore

(b) Caso di cache

(c) Caso peggiore

in base all'ipotesi di nessun ritardo causato dalla memoria nell'esecuzione delle istruzioni. Si presuppone che il tempo di accesso alla memoria sia di tre cicli di clock per un'operazione di lettura o di scrittura di 32 bit. La Tab. 13.7 elenca brevemente il numero di cicli di clock richiesti per operazioni selezionate in ciascuna delle tre categorie. Il *caso migliore* si presenta quando un'istruzione è nella memoria cache e beneficia della massima sovrapposizione temporale con altre istruzioni.[4] Se non c'è alcuna sovrapposizione ma l'istruzione è nel cache, allora si applica il *caso di cache* di temporizzazione. Nel *caso peggiore*, non c'è alcuna sovrapposizione e l'istruzione non si trova nella memoria cache oppure il cache è disabilitato. Si rimanda il lettore all'*MC68020 User's Manual* per ulteriori dettagli concernenti la temporizzazione delle istruzioni. La temporizzazione di un programma effettuata mediante programmi di benchmark sarà discussa nel par. 14.4.

Tab. 13.7 *Tempi di esecuzione delle istruzioni (MC68020).*

Operazione	**Numero di cicli di clock**		
	Caso migliore	Caso di cache	Caso peggiore
Lettura o scrittura della memoria	---	---	3
Calcolo dell'indirizzo effettivo e prelievo dell'operando	0-17	0-20	0-24
Esecuzione dell'istruzione (incluso il prelievo dell'istruzione)	0-88	2-90	3-90
Risposta all'interruzione	26	26	33
Istruzione di trappola	20	20	27

Nota: Alcune istruzioni richiedono più cicli di quelli delle istruzioni mostrate (ad esempio, RTE e RESET). Qualsiasi ritardo causato dalla durata del ciclo di memoria dev'essere aggiunto alla stima del tempo di esecuzione per ogni accesso alla memoria. Molte istruzioni e la risposta all'interruzione richiedono più accessi alla memoria.

[4] Il progetto interno dell'MC68020 è stato descritto nel par. 4.1. Un pipeline ed altre operazioni parallele nella CPU consentono la sovrapposizione delle istruzioni.

ESERCIZI

13.3.1 Si calcolino i tempi minimo e massimo in microsecondi per processori MC68020 operanti a 12.5, 16.67, 20 e 25 MHz in ciascuno dei seguenti casi.

(a) Esecuzione dell'istruzione (caso di cache)
(b) Risposta all'interruzione
(c) Istruzione TRAP

13.3.2 Qual è il tempo del caso peggiore per una risposta all'interruzione se è appena iniziato il prelievo di un'istruzione quando avviene l'interruzione per il processore a 20 MHz?

13.3.3 Si utilizzi l'istruzione di "nessuna operazione" (*No OPeration:* NOP) dellla CPU MC68020 per programmare un ciclo di ritardo con ritardi variabili da 100 microsecondi a 1 secondo. Il tempo di ritardo in microsecondi dovrebbe essere inserito dal terminale dall'operatore prima che inizi il ciclo di ritardo. NOP richiede due cicli di clock (caso di cache) o tre cicli di clock (caso peggiore). L'istruzione è progettata per impedire la sovrapposizione delle istruzioni, per cui anche l'operazione del caso migliore richiede due cicli di clock.

13.4 CONSIDERAZIONI SULL'HARDWARE

La connessione elettrica tra gli elementi in un sistema basato sull'MC68020 avviene tramite il bus interno di sistema.[5] Il bus contiene linee di segnali d'indirizzo, linee di segnali di dati, e linee di segnali di controllo. Sono incluse anche alcune linee di segnale miscellanee, come quelle per il clock, per l'alimentazione (+5 volt) ed una linea di riferimento di massa. Tutti i dispositivi periferici ed il processore sono connessi in parallelo ai segnali di bus. Nella maggior parte delle applicazioni, il processore MC68020 agisce da controllore del sistema e determina l'utilizzazione del bus per il trasferimento di dati e di segnali di controllo. Le linee di segnale dell'MC68020 sono mostrate nella Fig. 13.11. Esse sono definite in base alla funzione, al nome mnemonico ed alle caratteristiche nella Tab. 13.8. Le linee di segnale descritte come "a 3 stati" nella tabella sono poste in uno stato di alta impedenza (hi-Z) quando il processore rilascia il bus per altri dispositivi. Queste linee di segnale si comportano, a tutti gli effetti, come se fossero sconnesse elettricamente dal bus quando si trovano in tale stato.

Una comprensione dell'impiego delle linee di segnale dell'MC68020 per le operazioni fondamentali è necessario al fine di progettare le interfacce. In questo paragrafo sono descritte le operazioni di trasferimento di dati e l'uso delle linee di codice di funzione durante gli accessi del processore alla memoria. Queste informazioni sono particolarmente importanti per il progettista di sistema, quando si

[5] Il bus interno di sistema discusso in questo paragrafo è connesso direttamente alle linee di segnale della CPU. Nel cap. 14 sarà trattato il VMEbus utilizzato per connettere moduli di computer del tipo descritto nel par. 1.2.

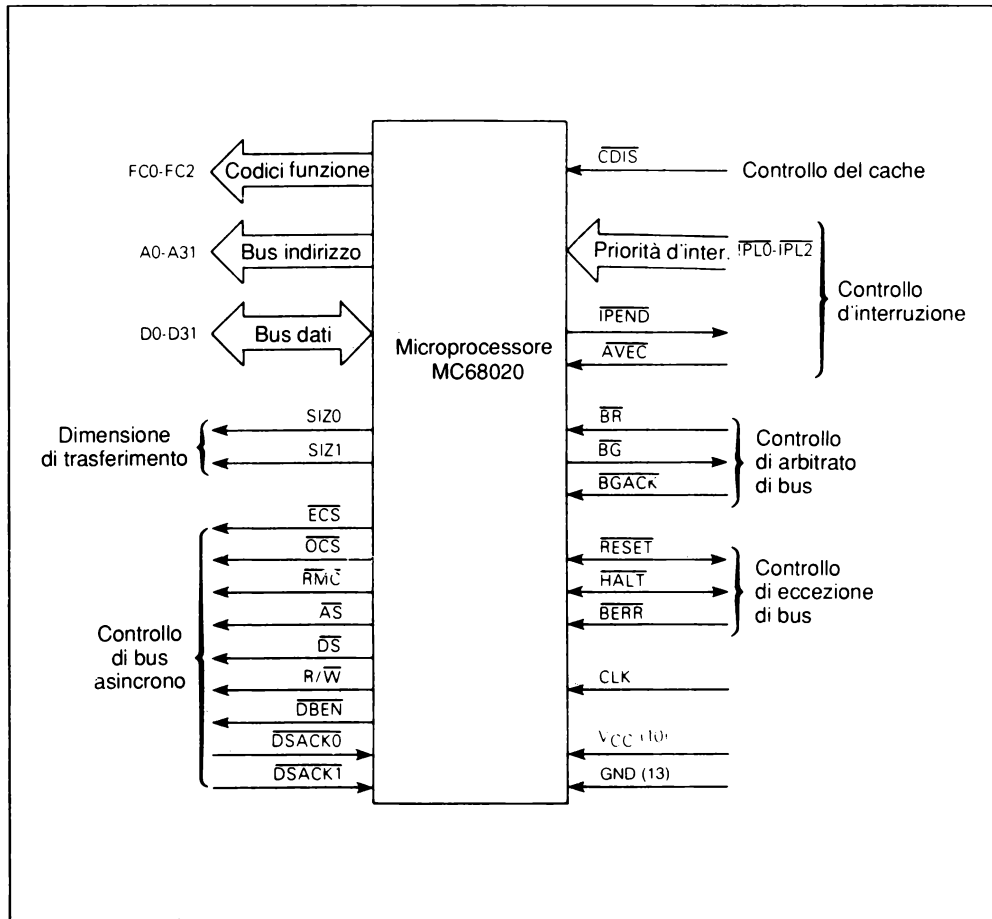



Fig. 13.11 *Linee di segnale dell'MC68020. (Per gentile concessione di Motorola, Inc.)*

adotta un metodo di gestione della memoria per proteggere e segmentare diverse aree di memoria. La descrizione comprende inoltre la sequenza d'interruzione hardware, il controllo del bus da parte di altri dispositivi e l'interfaccia di coprocessore.

Si dovrebbe notare che le informazioni fornite in questo paragrafo riguardano le operazioni funzionali delle linee di segnale dell'MC68020. Non saranno presentati i diagrammi di temporizzazione precisi necessari ad un progettista d'interfaccia. La fonte di questi diagrammi è costituita dai dati tecnici (*data sheets*) della Motorola per l'MC68020, che contengono le specifiche elettriche per la CPU. I valori massimi e minimi degli intervalli temporali per le variazioni di un certo segnale e gli intervalli tra le variazioni su diverse linee di segnali (utili per la determinazione dei margini di temporizzazione ammissibili) sono reperibili nelle specifiche tecniche incluse nell'*MC68020 User's Manual*.

Tab. 13.8 *Definizioni delle linee di segnale dell'MC68020.*

Funzione del segnale	Nome del segnale	Ingresso/uscita	Stato attivo	Tre stati
Codici di funzione (*Function Codes*)	FC0—FC2	Uscita	Alto	Sì
Bus d'indirizzo (*Address bus*)	A0—A31	Uscita	Alto	Sì
Bus di dati (*Data bus*)	D0—D31	Ingresso/Uscita	Alto	Sì
Dimensione (*Size*)	SIZ0—SIZ1	Uscita	Alto	Sì
Inizio del ciclo esterno (*External Cycle Start*)	$\overline{\text{ECS}}$	Uscita	Basso	No
Inizio del ciclo di operando (*Operand Cycle Start*)	$\overline{\text{OCS}}$	Uscita	Basso	No
Ciclo di lettura-modifica-scrittura (*Read-Modify-write Cycle*)	$\overline{\text{RMC}}$	Uscita	Basso	Sì
Abilitazione d'indirizzo (*Address Strobe*)	$\overline{\text{AS}}$	Uscita	Basso	Sì
Abilitazione di dati (*Data Strobe*)	$\overline{\text{DS}}$	Uscita	Basso	Sì
Lettura/scrittura (*Read/Write*)	R/$\overline{\text{W}}$	Uscita	Alto/Basso	Sì
Abilitazione del buffer di dati (*Data Buffer ENable*)	$\overline{\text{DBEN}}$	Uscita	Basso	Sì
Riconoscimento del trasferimento di dati e della dimensione (*Data transfer and Size ACKnowledge*)	$\overline{\text{DSACK0}}$/ $\overline{\text{DSACK1}}$	Ingresso	Basso	—
Disabilitazione del cache (*Cache DISable*)	$\overline{\text{CDIS}}$	Ingresso	Basso	—
Livello di priorità d'interruzione (*Interrupt Priority Level*)	$\overline{\text{IPL0}}$—$\overline{\text{IPL2}}$	Ingresso	Basso	—
Interruzione in sospeso (*Interrupt PENDing*)	$\overline{\text{IPEND}}$	Uscita	Basso	No
Autovettore (*AutoVECtor*)	$\overline{\text{AVEC}}$	Ingresso	Basso	—
Richiesta di bus (*Bus Request*)	$\overline{\text{BR}}$	Ingresso	Basso	—
Concessione di bus (*Bus Grant*)	$\overline{\text{BG}}$	Uscita	Basso	No
Riconoscimento di concessione di bus (*Bus Grant ACKnowledge*)	$\overline{\text{BGACK}}$	Ingresso	Basso	—
Reset (*RESET*)	$\overline{\text{RESET}}$	Ingresso/Uscita	Basso	No (pozzo aperto)
Arresto (*HALT*)	$\overline{\text{HALT}}$	Ingresso/Uscita	Basso	No (pozzo aperto)
Errore di bus (*Bus ERRor*)	$\overline{\text{BERR}}$	Ingresso	—	—
Clock (*CLocK*)	$\overline{\text{CLK}}$	Ingresso	—	—
Alimentazione (*power supply*)	V_{CC}	Ingresso	—	—
Massa (*GrouND*)	GND	Ingresso		—

Fonte: Motorola, Inc.

Prima di discutere le varie operazioni dell'hardware, devono essere presentate le convenzioni adottate per specificare le linee di segnale. Ulteriori dettagli sul progetto dell'hardware sono presentati in vari riferimenti bibliografici relativi a questo capitolo, elencati nell'app. E alla fine del libro.

Designazione delle linee di segnale. I segnali elettrici generati o ricevuti dall'MC68020 possono essere specificati in termini delle loro caratteristiche elettriche e anche dal loro impiego logico o funzionale. Tali caratteristiche elettriche includono il livello di tensione, i requisiti di corrente e la velocità di commutazione. Inoltre, devono essere prese in considerazione altre caratteristiche dei segnali generati dal processore o da un dispositivo esterno e che si propagano lungo le linee di segnale del bus di sistema.

Il processore e i dispositivi esterni rispondono ad un segnale in conformità con lo stato del segnale. Lo stato per una linea con due stati può essere designato come attivo/inattivo, ALTO/BASSO, vero/falso, oppure {1}/{0}. Queste quattro designazioni sono considerate equivalenti quando s'impiegano la compatibilità TTL (*Transistor-Transistor Logic:* logica transistore-transistore) e la logica positiva.

Tutte le linee di segnale dell'MC68020 sono TTL-compatibili, per cui il processore può essere connesso elettricamente a qualsiasi chip TTL purché siano rispettate certe condizioni. Poiché la famiglia TTL è il tipo di logica attualmente più diffuso per la circuiteria d'interfacciamento, il processore MC68020 può essere facilmente connesso ad un gran numero di circuiti TTL che sono disponibili per implementare varie operazioni logiche. Una linea di segnale TTL è considerata essere nello stato BASSO quando la sua tensione stazionaria è 0 volt rispetto al riferimento di massa. Lo stato ALTO di un segnale TTL è indicato da un livello di tensione di 5 volt rispetto alla massa. La tensione di alimentazione di un sistema TTL, designata come V_{CC}, ha un valore tipico (nominale) di 5.0 volt. Qualsiasi linea TTL designata dal suo nome funzionale (di solito, un codice mnemonico) è considerata "attiva" o "vera" o al livello {1} logico quando il livello di tensione è ALTO. Viceversa, la linea è "inattiva" o "falsa" o al livello {0} logico quando il livello di tensione è BASSO. Qualunque segnale, designato come la negazione logica (NOT) della sua funzione, rappresenta le condizioni opposte. In questo caso, una tensione BASSA rappresenta uno stato attivo, o vero, o {1}; tutti i segnali di questo tipo (negati) saranno designati mediante una sbarretta di sopralineatura (che è il simbolo del NOT logico).

Quindi le linee di dati dell'MC68020 sono designate come D0, D1, ..., D7, in cui uno stato TTL ALTO su qualsiasi linea rappresenta {1}, mentre uno stato TTL BASSO rappresenta {0}. D'altro canto, la linea di segnale $\overline{AS}$ (*Address Strobe:* abilitazione dell'indirizzo) in uno stato TTL ALTO indica che l'abilitazione dell'indirizzo non è attivata (cioè, non è attiva né vera). Quando la tensione è bassa sulla linea $\overline{AS}$, la linea di abilitazione dell'indirizzo è attivata (attiva o vera). Questo impiego di un segnale attivo BASSO è alquanto comune con la logica TTL ed offre una migliore immunità al rumore elettrico in certe condizioni. Una linea di segnale siffatta è spesso denotata come "attiva bassa" e si parla di "AS negato". In un libro stampato, tale distinzione non è necessaria, poiché la forma $\overline{AS}$ indica già l'operazione logica della linea di segnale.

13.4.1 Operazioni di trasferimento di dati

Sono qui presentate le operazioni di lettura e di scrittura del processore MC68020. Un'operazione di lettura richiede che i dati da un dispositivo esterno o dalla memoria siano posti sul bus in risposta alle linee di segnale controllate dall'MC68020. Durante un'operazione di scrittura i processore presenta sul bus i dati da registrare nella memoria o da inviare in uscita a un dispositivo esterno. Il processore controlla il bus per avviare operazioni di trasferimento di dati nell'una o nell'altra direzione, ma attende che il dispositivo selezionato riconosca la richiesta di trasferimento. Tali operazioni di trasferimento di dati sono denominate *asincrone* poiché è la temporizzazione della memoria o di un dispositivo periferico, anziché quella della CPU, che determina la temporizzazione del trasferimento.[6] Di solito, questi dispositivi sono più lenti rispetto al processore per il trasferimento dei dati.

Il ciclo di lettura. La Fig. 13.12 illustra la sequenza degli eventi e la temporizzazione per un ciclo di lettura. Per esempio, se viene eseguita un'istruzione della forma:

```
MOVE.L    MEMORIA,D1
```

allora la CPU esegue un ciclo di lettura di longword dalla locazione di memoria specificata dall'etichetta MEMORIA.[7] La linea di segnale di lettura/scrittura (*Read/ Write:* R/$\overline{\text{W}}$) e le linee di segnale del codice di funzione indicano il tipo di trasferimento di dati. Le linee dei segnali d'indirizzo indicano la locazione nella memoria. Queste linee sono considerate attive o valide durante i primi due stati S0 e S1 del clock nella Fig. 13.12(b). Le linee dei segnali di dimensione (SIZ1, SIZ0) sono entrambe BASSE per indicare che la CPU deve leggere un valore di 32 bit.

I circuiti di controllo della memoria devono presentare il valore del dato sulle linee dei segnali di dati, dopo che i segnali $\overline{\text{AS}}$ (*Address Strobe:* abilitazione dell'indirizzo) e $\overline{\text{DS}}$ (*Data Strobe:* abilitazione del dato) sono stati condotti nello stato BASSO dalla CPU.[8] Quando le linee di segnale sono valide, i circuiti di controllo della memoria devono attivare le linee dei segnali di riconoscimento del trasferimento di dati e della dimensione (*Data transfer and Size ACKnowledge:* $\overline{\text{DSACK0}}$, $\overline{\text{DSACK1}}$), che indicano la dimensione (8, 16 o 32 bit) della porta della memoria. Nell'esempio mostrato, entrambe le linee di segnale sono tenute BASSE, per designare un trasferimento di 32 bit.

---

[6] Operazioni sincrone basate sul clock della CPU sono possibili se un'opportuna circuiteria esterna viene aggiunta al sistema.

[7] Poiché l'MC68020 impiega l'I/O rappresentato nella memoria, questo indirizzo potrebbe individuare il registro di un chip periferico anziché la memoria.

[8] Vari altri segnali ($\overline{\text{ECS}}$, $\overline{\text{OCS}}$ e $\overline{\text{DBEN}}$) possono essere usati per avviare il trasferimento di dati da parte del dispositivo esterno. L'impiego di questi segnali è descritto nell'*MC68020 User's Manual.*

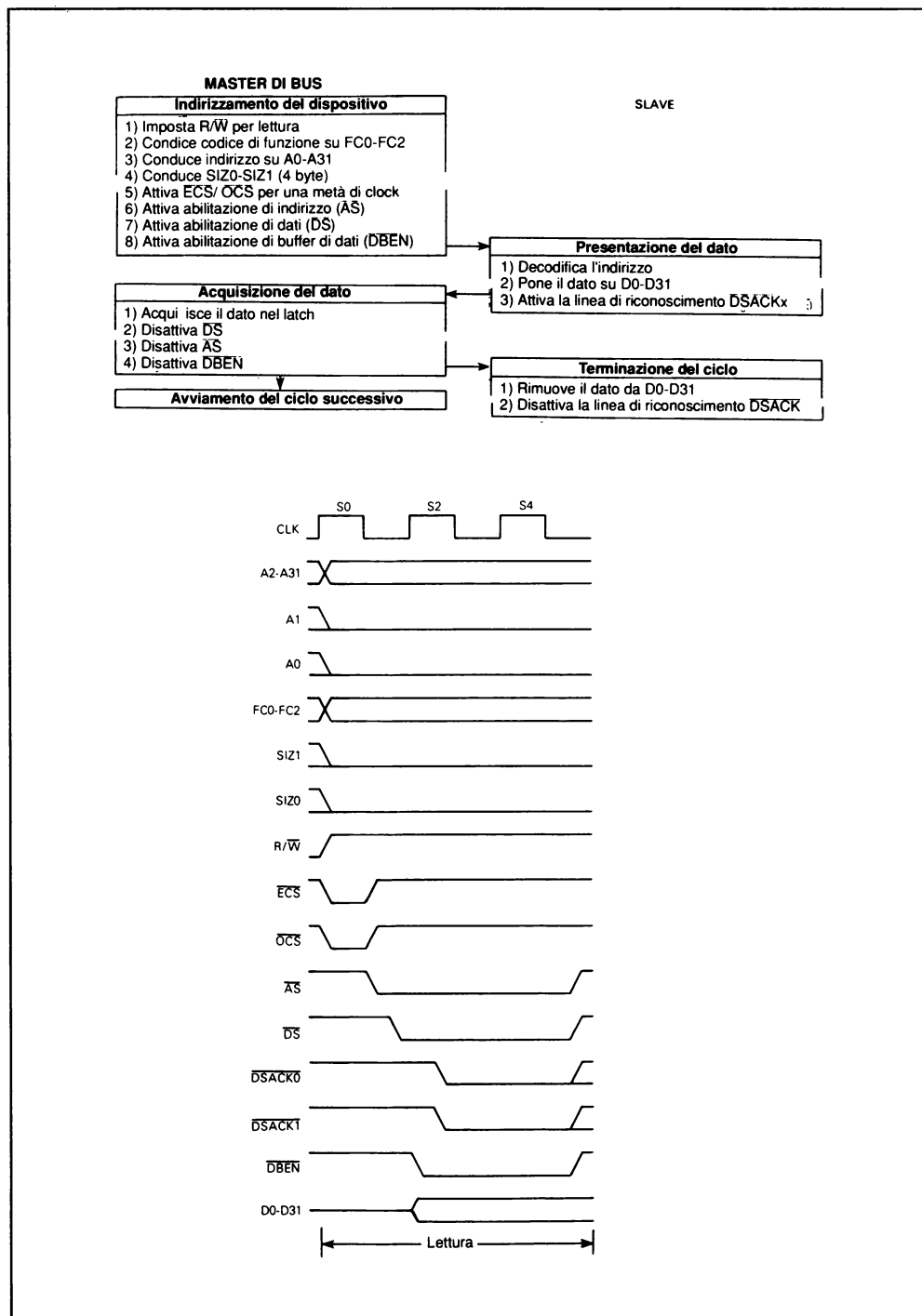



Fig. 13.12 *(a) Sequenza per una lettura di longword. (b) Temporizzazione per una lettura di longword. (Per gentile concessione di Motorola, Inc.)*

La CPU acquisisce internamente i dati e notifica questo fatto alla memoria, negando (cioè, ponendo nello stato ALTO) $\overline{AS}$ e $\overline{DS}$. Un ciclo di lettura è terminato dopo lo stato quattro (S4), allorché la memoria nega le sue linee di segnale $\overline{DSACK0}$ e $\overline{DSACK1}$. Se non ci sono ritardi causati dai circuiti di memoria, l'intero trasferimento richiede tre cicli di clock, definiti dagli stati S0-S5.

Se i circuiti di memoria non sono in grado di rispondere alla richiesta di lettura entro la fine dello stato due (S2), la CPU genera uno stato di "attesa" della durata di un ciclo di clock. La CPU continuerà a generare stati di attesa finché le memorià non avrà risposto. Uno speciale dispositivo esterno, quale un timer "watchdog" (letteralmente: cane da guardia) dev'essere presente nel sistema per terminare il ciclo di bus, nell'eventualità che la memoria non possa rispondere. Dopo il tempo concesso, il timer genera una condizione di errore di bus, per indicare che la locazione di memoria è difettosa o non è presente all'indirizzo emesso dalla CPU. La risposta all'errore di bus sarà discussa nel sottopar. 13.4.5.

Il ciclo di scrittura. Un ciclo di scrittura della CPU è simile al ciclo di lettura sia per la sequenza che per la temporizzazione. La linea di segnale di lettura/scrittura è mantenuta BASSA per indicare un'operazione di scrittura. Il processore non rimuove i dati validi dalle linee dei segnali di dati finché un dispositivo esterno o la memoria non avrà risposto con $\overline{DSACK0}$ e $\overline{DSACK1}$. Tre cicli di clock sono il tempo minimo richiesto per un ciclo di scrittura.

Dimensione della porta periferica. Le linee dei segnali di dati del processore MC68020 possono essere impiegate per trasferire dati tra la CPU e un dispositivo esterno, anche se tale dispositivo non è in grado di utilizzare tutti i 32 bit del bus di dati della CPU. La *dimensione della porta* del dispositivo è definita come il numero di bit di dati che esso può trasferire in un'unica operazione. Nei sistemi basati sull'MC68020, sono ammesse dimensioni di porta di 8, 16 e 32 bit. Dispositivi di dimensioni diverse devono essere connessi al bus dati come mostrato nella Fig. 13.13. La figura mostra un trasferimento di 32 bit dalla CPU alle porte di lunghezza di byte, word o longword. Si noti che una porta la cui dimensione sia inferiore a 32 bit dev'essere connessa ala porzione superiore del bus di dati. Se la connessione dell'hardware è corretta, il programmatore non dovrà interessarsi della configurazione dell'hardware. Ad esempio, il trasferimento

```
MOVE.L    D1,REGBYTE
```

causerebbe il trasferimento di quattro byte in quattro cicli di scrittura alla locazione REGBYTE se il dispositivo a quell'indirizzo ha una porta di 8 bit. Le linee dei segnali $\overline{DSACK0}$ e $\overline{DSACK1}$ sono impiegate dal dispositivo per indicare la sua dimensione di porta.

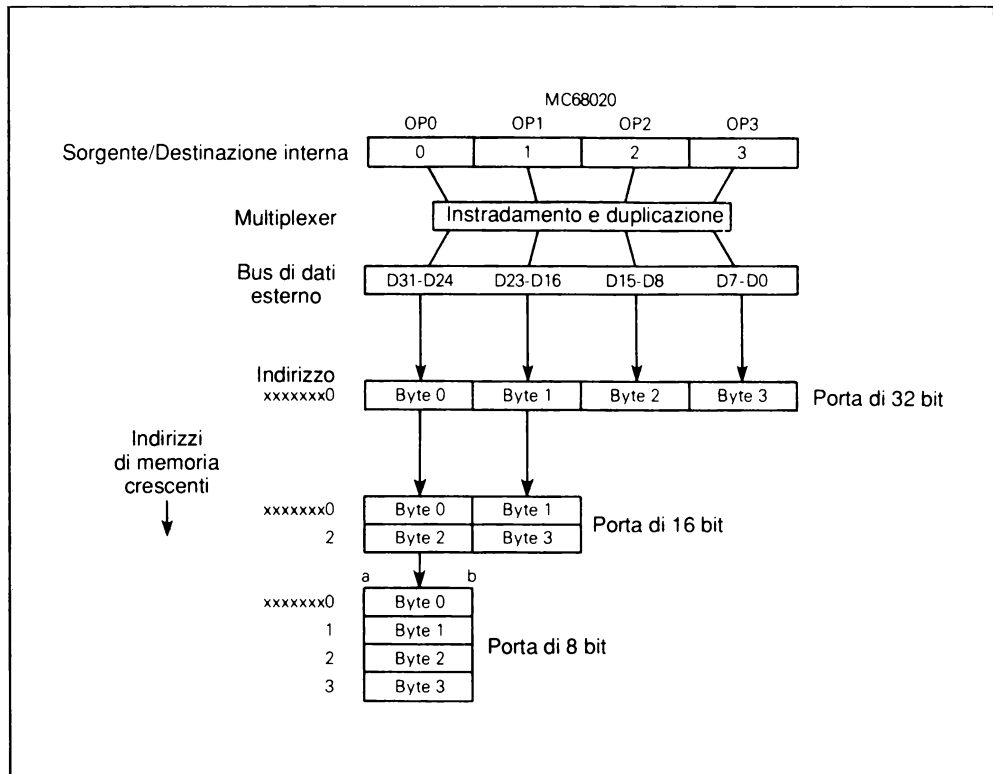



Fig. 13.13 *Interfaccia dell'MC68020 con varie dimensioni di porta. (Per gentile concessione di Motorola, Inc.)*

13.4.2 Dimensionamento dinamico del bus e trasferimento disallineato

L'MC68020 consente di trasferire operandi di byte, word o longword da o verso porte di 8, 16 o 32 bit. Un'istruzione in corso di esecuzione determina la dimensione dell'operando, indipendentemente dalla dimensione della porta. Comunque, le linee dei segnali di riconoscimento del trasferimento di dati e della dimensione ($\overline{\text{DSACKx}}$) sono impiegate per specificare la dimensione della porta per ogni ciclo di lettura o scrittura. Questa capacità dell'MC68020 di trasferire i dati tra porte di dimensioni differenti è nota come *dimensionamento dinamico del bus* e rende flessibile il progetto dell'hardware. Inoltre, il programmatore può trasferire operandi di qualsiasi dimensione senza preoccuparsi dei dettagli dell'interfaccia per i trasferimenti di dati.

Gli operandi nella memoria possono iniziare da qualsiasi indirizzo di byte, senza tener conto della dimensione dell'operando. Quindi gli operandi di word o di longword possono essere situati a indirizzi di memoria dispari. Poiché la CPU può aver bisogno di eseguire più cicli di bus per trasferire tali operandi, indipendente-

mente dalla dimensione della porta, i trasferimenti sono considerati *disallineati.*[9] Non è necessaria alcuna particolare programmazione o progetto d'interfaccia speciale per effettuare trasferimenti disallineati in un sistema basato sull'MC68020.

13.4.3 Linee del codice di funzione ed utilizzazione della memoria

Le linee dei segnali FC0, FC1 e FC2 presentano un codice di funzione ai dispositivi esterni o alla memoria, che indica il tipo di attività in corso sulle linee dei segnali di indirizzo o di dati. Il codice di funzione indica il modo del processore (supervisore o utente) ed il tipo di accesso (dati o programma) ogni volta che il processore inizia un'operazione di lettura o di scrittura. Il codice indica anche se un'interruzione è in corso di riconoscimento o se un coprocessore sta effettuando un indirizzamento.

La Tab. 13.9 elenca lo stato elettrico delle linee dei segnali del codice di funzione dell'MC68020 per ciascun tipo di riferimento. Gli stati non specificati non sono definiti. Il modo di processore è sempre determinato dal valore del bit di stato del supervisore nel registro di stato, (SR)[13]. La distinzione tra riferimenti di dati e di programma è determinata dalla modalità d'indirizzamento e dall'istruzione in corso di esecuzione. La Tab. 13.10 definisce le modalità d'indirizzamento che causano la selezione di vari stati del codice di funzioni durante la normale esecuzione.

Tab. 13.9 *Riferimenti di codice di funzione.*

FC2	FC1	FC0	Tipo di ciclo
0	0	0	(Indefinito, riservato)
0	0	1	Spazio di dati di utente
0	1	0	Spazio di programma di utente
0	1	1	(Indefinito, riservato)
1	0	0	(Indefinito, riservato)
1	0	1	Spazio di dati di supervisore
1	1	0	Spazio di programma di supervisore
1	1	1	Spazio di CPU

Nota: Lo spazio d'indirizzi 3 è riservato per la definizione di utente, mentre 0 e 4 sono riservati per usi futuri dalla Motorola.

Fonte: Motorola, Inc.

[9] Come descritto nel par. 4.6, le istruzioni devono essere allineate su confini di word (byte pari, cioè i loro indirizzi di memoria devono essere pari)

Tab. 13.10 *Modalità d'indirizzamento e codici di funzione.*

Riferimento del codice di funzione	Modalità d'indirizzamento
Dati	Tutti i modi indiretti, a meno che non siano usati con le istruzioni JMP o JSR. Modi assoluti, a meno che non siano usati con le istruzioni JMP o JSR.
Programma	Modi relativi Modi indiretti e assoluti con le istruzioni JMP e JSR.

Fonte: Motorola, Inc.

Usando le linee del codice di funzione per selezionare le aree di memoria, la memoria del sistema può essere segmentata in spazio di supervisore e spazio di utente. Questi spazi, a loro volta, possono essere segmentati in aree di programma e aree di dati. Un metodo semplificato per effettuare tale segmentazione è mostrato nella Fig. 13.14. Ogni codice di funzione distinto consente di accedere ad un blocco di memoria selezionato, o ad un gruppo di blocchi, mediante un decodificatore ed una circuiteria di selezione. Si può progettare un meccasimo più avanzato di protezione della memoria, compresa la protezione dalla scrittura di certe aree di memoria, impiegando il chip di gestione della memoria prodotto dalla Motorola. Questo chip separa gli spazi di utente e di supervisore e protegge la memoria in base alle linee dei segnali d'indirizzo.

I registri DFC e SFC e l'istruzione MOVES. Nel par. 10.1 sono stati presentati il registro del codice di funzione di destinazione (*Destination Function Code:* DFC) ed il registro del codice di funzione di sorgente (*Source Function Code:* SFC). In quel paragrafo, questi registri speciali accessibili soltanto da un programma nel modo di supervisore, sono stati definiti come parte del modello di programmazione di supervisore. Ciascun registro è di 32 bit, sebbene i 29 bit superiori siano letti come zeri ed ignorati quando vengono scritti. L'istruzione MOVEC (*MOVE Control register:* trasferisci registro di controllo), descritta nel par. 10.2, è usata per leggere il valore del registro o scrivere un nuovo valore nel DFC o SFC.

I registri del codice di funzione alternativo (DFC e SFC) contengono i valori dello spazio d'indirizzi posti in FC0-FC2 durante la lettura o la scrittura dell'operan-

Note:

(1) Il decodificatore attiva una singola linea, che corrisponde al valore binario dei bit del codice di funzione.

(2) La logica di selezione abilita i blocchi di memoria selezionati. Essa potrebbe essere utilizzata per consentire ad un programma di supervisore di accedere a tutta la memoria, per esempio, tuttavia limita l'accesso di utente al blocco appropriato.

(3) La logica di errore è necessaria per indicare un errore, qualora venga effettuato un accesso nel blocco sbagliato.

(4) Non sono mostrate le linee di segnale di indirizzo, di dati e di controllo verso la memoria.

Fig. 13.14 *Protezione della memoria.*

do di un'istruzione MOVES (*MOVE address Space:* trasferisce spazio di indirizzi). Tali registri sono quindi usati da un programma operante nel modo di supervisore per modificare temporaneamente i valori del codice di funzione mentre viene eseguita l'istruzione MOVES. Normalmente, il programma nel modo di supervisore fa sì che il codice di funzione appropriato per lo spazio di utente sia inviato in uscita se la memoria è protetta in base alle modalità di supervisore e di utente. In un sistema protetto, il programma di supervisore può accedere a tutta la memoria usando l'istruzione MOVES. In sistemi che non dispongono di hardware di gestione della memoria che limita l'accesso di supervisore allo spazio d'indirizzi di utente, tale manipolazione dei valori del codice di funzione non è necessaria. L'istruzione MOVES può essere usata anche per scrivere nello spazio di programma se la memoria è protetta dall'accesso del programma e dei dati.

Esempio 13-2

La subroutine MTVUSER nella Fig. 13.15 trasferisce un array di valori di 16 bit dall'area di memoria di supervisore allo spazio di utente. Prima che la subroutine venga chiamata, A0 deve contenere l'indirizzo iniziale dell'array nello spazio di supervisore. L'indirizzo di destinazione dev'essere contenuto in A1. La lunghezza dell'array è specificata in D0.

Nel ciclo per trasferire i valori, l'istruzione MOVEC definisce il contenuto del registro del codice di funzione di destinazione (DFC) per indicare lo spazio di utente nella memoria quando viene eseguita l'istruzione MOVES. Tranne che per i trasferimenti effettuati da MOVES, tutti gli altri riferimenti alla memoria riguardano lo spazio di supervisore. Pertanto, la subroutine dev'essere eseguita nel modo di supervisore della CPU. Quando i valori di dati sono stati trasferiti, i valori salvati del registro sono ripristinati dallo stack di sistema e il controllo viene restituito al programma chiamante.

```
                              1.           TTL      FIGURA 13.15
                              2.           LLEN     100
00007600                      3.           ORG      $7600
                              4.  *
                              5.  *   TRASFERIMENTO DI UN ARRAY DI WORD DALLO SPAZIO
                              6.  *   DI SUPERVISORE ALLO SPAZIO DI DATI DI UTENTE
                              7.  *
                              8.  *        INPUT:  (A0.L) = INDIRIZZO DELL'ARRAY DI SORGENTE
                              9.  *                (A1.L) = INDIRIZZO DELL'ARRAY
                             10.  *                         DI DESTINAZIONE
                             11.  *                (D0.L) = NUMERO DI WORD NELL'ARRAY
                             12.  *
                             13.  *        OUTPUT: ARRAY NELLO SPAZIO DI DATI DI UTENTE
                             14.  *
    # 00000001               15.  USERDAT EQU      $0001              ;SPAZIO DI DATI DI UTENTE
00007600 48E7 F0C0           16.  MVTUSER MOVEM.L  A0-A1/D0-D3,-(A7)
00007604 7201                17.          MOVE.L   $USERDAT,D1
00007606 4283                18.          CLR.L    D3                 ;REGISTRO INDICE ZERO
00007608 4E7B 1001           19.  LOOP    MOVEC    D1,DFC             ;CAMBIA IN UTENTE
0000760C 3418                20.          MOVE.W   (A0)+,D2
0000760E 0E59 2800           21.          MOVES.W  D2,(A1)+           ;TRASFERISCE
00007612 0683 00000001       22.          ADDI.L   #1,D3              ;INCREMENTA L'INDICE
00007618 B083                23.          CMP.L    D3,D0              ;SE (D3) < (D0)
0000761A 62 EC               24.          BHI      LOOP               ;  TRASFERISCE
                             25.  *                                   ;  ALTRIMENTI, RITORNA
0000761C 4CDF 030F           26.          MOVEM.L  (A7)+,A0-A1/D0-D3
00007620 4E75                27.          RTS
                             28.
```

Fig. 13.15 *Esempio di istruzione MOVES.*

Lo spazio di CPU. I normali cicli di bus del processore fanno riferimento ad aree di programma o ad aree di dati nella memoria, secondo la definizione dei valori del codice di funzione nella Tab. 13.9. Una terza categoria di riferimenti riguarda lo spazio di CPU di un sistema basato sull'MC68020. Lo spazio viene distinto dai valori del codice di funzione (111), come mostrato nella Fig. 13.16. I bit del bus d'indirizzo A16-A19 distinguono quattro tipi di attività dello spazio di CPU.

	Codice di funzione (2–0)	Bus d'indirizzo (31–0)
Riconoscimento di breakpoint	1 1 1	0 0 0 0 0 0 0 0 0 0 0 0 | 0 0 0 0 | 0 0 0 0 0 0 0 0 0 0 0 | N. BKPT | 0 0
Controllo del livello di accesso	1 1 1	0 0 0 0 0 0 0 0 0 0 0 0 | 0 0 0 1 | 0 0 0 0 0 0 0 0 0 | REG. MMU
Comunicazioni di coprocess.	1 1 1	0 0 0 0 0 0 0 0 0 0 0 0 | 0 0 1 0 | CPID | 0 0 0 0 0 0 0 0 | REG. CP
Riconoscimento d'interruzione	1 1 1	1 1 1 1 1 1 1 1 1 1 1 1 | 1 1 1 1 | 1 1 1 1 1 1 1 1 1 1 1 1 | LIVELLO | 1

Campo del tipo di spazio di CPU (bit 19–16)

Fig. 13.16 *Indirizzamento nello spazio di CPU.*

Un ciclo di riconoscimento del breakpoint si ha quando viene eseguita l'istruzione BKPT (*BreaKPoinT*). Un'istruzione CALLM causa un riferimento di controllo del livello di accesso nello spazio di CPU, come descritto nell'*MC68020 User's Manual*. Il ciclo di riconoscimento dell'interruzione ed il ciclo di comunicazione del coprocessore saranno descritti in seguito.

13.4.4 Elaborazione delle interruzioni

Nel par. 11.6 sono state svolte alcune considerazioni di programmazione per routine di gestione dell'interruzione. In tale paragrafo, sono state spiegate la maschera d'interruzione del registro di stato (SR[10:8]) e le locazioni del vettore per le routine d'interruzione. La circuiteria d'interruzione dell'MC68020 consente sette livelli di priorità d'interruzione, numerati da 1 a 7. Tutti i livelli tranne il settimo, che è quello dell'interruzione di priorità più elevata, possono essere mascherati o disabilitati usando la maschera d'interruzione del registro di stato. La discussione presente descrive le linee di segnale interessate nell'elaborazione delle interruzioni.

Una richiesta d'interruzione viene sottoposta alla CPU codificando il livello di tale richiesta sulle tre linee di segnale di richiesta d'interruzione ($\overline{IPL0}$, $\overline{IPL1}$, $\overline{IPL2}$). Un dispositivo esterno codifica il livello di priorità su queste linee di segnale come un valore binario:

$$\text{livello} = \overline{IPL2} \times 2^2 + \overline{IPL1} \times 2^1 + \overline{IPL0}$$

dove un segnale BASSO è considerato {1}, mentre un segnale ALTO è considerato {0} in questa equazione. Un livello zero indica che al momento non c'è alcuna richiesta d'interruzione. Affinché una richiesta d'interruzione venga riconosciuta, il suo livello, da 1 a 6, dev'essere maggiore del livello di maschera definito nel registro di stato. Il livello 7 rappresenta un'interruzione non mascherabile, che non può essere disabilitata.

La CPU risponde ad un'interruzione al completamento del ciclo d'istruzione corrente, se non c'è alcuna eccezione di priorità superiore in sospeso. Se l'interruzione è valida, la CPU attiva la linea di segnale $\overline{\text{IPEND}}$ (*Interrupt PENDing:* interruzione in sospeso), portandola nello stato BASSO, per indicare ai dispositivi esterni che l'MC68020 sta provvedendo a soddisfare una richiesta d'interruzione. La CPU deve quindi determinare la locazione iniziale della routine di servizio dell'interruzione per il livello d'interruzione richiesto. Il dispositivo richiedente può indicare il proprio numero di vettori in due modi. Nel modo *vettorizzato*, il dispositivo esterno fornisce il suo numero di vettore in risposta al ciclo di riconoscimento dell'interruzione della CPU. La CPU consente anche un modo *autovettorizzato*, in cui seleziona l'appropriata locazione autovettorizzata nella tabella di vettori, in base al livello dell'interruzione richiesta. Questi due modi sono discussi di seguito; inoltre sarà trattata la risposta della CPU ad un'interruzione spuria.

Interruzioni vettorizzate. La Fig. 13.17(a) descrive la sequenza di operazioni che hanno luogo quando un'interruzione viene accettata dalla CPU. Nella Fig. 13.17(b) è rappresentato il diagramma di temporizzazione corrispondente. La sequenza inizia quando la richiesta d'interruzione viene accettata dopo il completamento dell'istruzione corrente. La CPU accede al suo spazio di CPU, con le linee dei segnali d'indirizzo A19-A16 che indicano un ciclo di riconoscimento dell'interruzione. Le linee dei segnali d'indirizzo A1-A3 indicano il livello d'interruzione. Le linee di dimensione e $R/\overline{W}$ indicano che la CPU sta richiedendo un valore di dati di 8 bit, che rappresenta il numero del vettore del dispositivo interrompente. Ciò avviene nel primo ciclo di clock del ciclo di riconoscimento d'interruzione nella Fig. 13.17(b).

Un dispositivo richiedente che impiega la modalità operativa vettorizzata pone il suo numero di vettore nel byte meno significativo delle sue linee di segnali di dati. Quindi un dispositivo con una porta di 8 bit utilizza le linee D24-D31, mentre un dispositivo con una porta di 32 bit pone il suo numero di vettore in D0-D7. Questo numero indica uno dei 192 vettori d'interruzione di utente nella tabella dei vettori di eccezione (locazioni da $0100 a $03FC), come descritto nel par. 11.6. Dopo che la CPU ha ricevuto il numero del vettore, inizia l'elaborazione dell'interruzione e le informazioni appropriate vengono salvate nello stack. Dopodiché, sarà eseguita la routine di gestione dell'interruzione, a partire dall'indirizzo reperito nella tabella di vettori.

I 192 possibili vettori sono distribuiti sui sette livelli di priorità per le interruzioni, in base al progetto della circuiteria esterna. Qualunque sottopriorità per ciascuno dei sette livelli di CPU dev'essere determinata da circuiti esterni alla CPU stessa.

PROCESSORE

Riconoscimento dell'interruzione

1) Confronta il livello di richiesta d'interruzione con la maschera d'interruzione.
2) Imposta R/$\overline{W}$ per lettura.
3) Imposta il codice di funzione per spazio di CPU.
4) Pone il livello d'interruzione su A1, A2 e A3. Campo di tipo = IACK.
5) Imposta la dimensione per byte.
6) Attiva l'abilitazione dell'indirizzo ($\overline{AS}$) e l'abilitazione dei dati ($\overline{DS}$).

Acquisizione del numero del vettore

1) Acquisisce il numero di vettore nel latch.
2) Disattiva $\overline{DS}$ e $\overline{AS}$.

Inizio dell'elaborazione dell'interruzione

DISPOSITIVO INTERROMPENTE

Richiesta d'interruzione

Informazioni sul vettore

1) Pone il numero del vettore nel byte meno signif. della porta di dati (in base alla dim. della porta).
2) Attiva $\overline{DSACKx}$

— oppure —

1) Attiva AVEC per la generazione automatica del numero del vettore.

Rilascio

1) Disattiva $\overline{DSACKx}$.

S0 S2 S4 S0 S2 S4 S0 S2

CLK

A4-A31

A1-A3 Livello d'interruzione

A0

FC0-FC2

SIZ1

SIZ0

R/$\overline{W}$

$\overline{ECS}$

$\overline{OCS}$

$\overline{AS}$

$\overline{DS}$

$\overline{DSACK0}$

$\overline{DSACK1}$

$\overline{DBEN}$

Numero di vettore da porta di 8 bit

D24-D31

Numero di vettore da porta di 16 bit

D16-D23

Numero di vettore da porta di 32 bit

D0-D7

$\overline{IPL0}$-$\overline{IPL2}$

Riconoscimento dell'interruzione

Scrittura in stack

Fig. 13.17 *(a) Sequenza di riconoscimento di un'interruzione. (b) Temporizzazione del ciclo di riconoscimento d'interruzione. (Per gentile concessione di Motorola, Inc.)*

Interruzioni autovettorizzate. Una modalità operativa alternativa è il modo *autovettorizzato*. Esso richiede una circuiteria esterna meno complessa, poiché il dispositivo richiedente non deve fornire un numero di vettore. Tale dispositivo richiede l'autovettorizzazione attivando (ponendo nello stato BASSO) la linea di segnale $\overline{AVEC}$ della CPU durante il ciclo di riconoscimento dell'interruzione. Le sette interruzioni autovettorizzate hanno vettori in locazioni fisse nella tabella di vettori di eccezione (da \$064 a \$07C). Il numero di autovettore (1-7) corrisponde al livello di priorità della CPU.

Interruzione spuria. Durante un ciclo di riconoscimento d'interruzione, un dispositivo esterno deve rispondere con un numero di vettore, o con $\overline{AVEC}$ per l'autovettorizzazione, affinché il ciclo possa essere completato. Se nessun dispositivo risponde e la circuiteria esterna avanza una richiesta di errore di bus (*Bus ERRor:* $\overline{BERR}$), la CPU causa un'eccezione di interruzione spuria, anziché un'eccezione di errore di bus, come descritto nel par. 11.5. La linea di segnale di richiesta di errore di bus ($\overline{BERR}$) viene solitamente attivata da un timer *watchdog* per indicare che nessun dispositivo ha risposto al ciclo di riconoscimento dell'interruzione entro il tempo concesso.

13.4.5 Arbitrato di bus, RMC, errore di bus, alt e reset

Sono discussi qui due speciali cicli di bus, che controllano l'attività della CPU e del bus di sistema in vari modi. Un ciclo di *arbitrato di bus* è usato quando un dispositivo esterno o un'altra CPU assume il controllo del bus di sistema. In un sistema multiprocessore, un ciclo di lettura-modifica-scrittura (*Read-Modify-write Cycle:* RMC) è utilizzato per indicare la validità di un flag condiviso o di un semaforo che può essere modificato da vari processori. Questi cicli di bus speciali sono discussi nei prossimi due sottoparagrafi. In questo paragrafo sarà spiegato anche lo scopo di varie linee di segnale della CPU utilizzate in condizioni speciali.

Quando si presenta un problema durante un ciclo di bus della CPU, un dispositivo esterno utilizza la linea di segnale di richiesta di errore di bus ($\overline{BERR}$) per indicare il verificarsi di una condizione insolita. Se due errori gravi si presentano durante certi cicli di bus della CPU, una condizione di arresto (*halt*) causa la cessazione dell'elaborazione da parte della CPU. La linea di segnale di reset ($\overline{RESET}$) serve ad inizializzare la CPU ed altri dispositivi nel sistema.

Arbitrato di bus. Nei più semplici sistemi con un singolo processore, la CPU MC68020 controlla il bus di sistema mentre preleva ed esegue le istruzioni. La CPU determina l'attività sul bus di sistema presentando segnali di controllo e indirizzi sulle linee d'indirizzo. Tutti gli altri dispositivi connessi al bus di sistema rispondono ai segnali di controllo della CPU per effettuare i trasferimenti di I/O ed altre operazioni. Tuttavia, esistono molti sistemi di computer in cui un altro dispositivo deve assumere il controllo del bus di sistema per svolgere qualche operazione, indipendentemente dal processore che normalmente controlla il bus. In questi

sistemi, la CPU che normalmente controlla il bus di sistema lo deve rilasciare quando un altro dispositivo lo richiede. Ciò avviene elettricamente quando la CPU pone nello stato di alta impedenza le sue linee di segnali di indirizzi, di dati e di controllo, in risposta ad un segnale di richiesta di bus emesso da un altro dispositivo. Il metodo per determinare il dispositivo che controlla il bus durante un qualunque ciclo di bus è noto come *arbitrato di bus*.

L'esempio più comune della necessità dell'arbitrato di bus si presenta nei computer che includono dispositivi con capacità di accesso diretto alla memoria (*Direct Memory Access:* DMA). Tali dispositivi assumono il controllo del bus di sistema per eseguire i trasferimenti di I/O ad alta velocità tra la memoria ed una periferica. Un altro esempio della necessità dell'arbitrato di bus è stato presentato nel par. 12.6, a proposito della discussione dei sistemi multiprocessore. In tali sistemi, due o più processori condividono il bus di sistema.

Il processore o il dispositivo DMA che controlla il bus di sistema in un certo istante è denominato *master* (padrone) del bus. Quando un altro dispositivo, come uno di DMA, assume il controllo del bus di sistema, esso agisce come un master di bus "temporaneo". Quando il bus è condiviso da una sola CPU e da un solo dispositivo DMA, le linee di segnale di arbitrato di bus dell'MC68020 sono sufficienti per svolgere la propria funzione. Se vari dispositivi o una seconda CPU condividono il bus con l'MC68020, allora è necessaria una circuiteria speciale per l'arbitrato di bus. Tale circuiteria serve a determinare la priorità delle richieste, per evitare i conflitti che sorgerebbero qualora più dispositivi richiedessero il bus simultaneamente.

La Fig. 13.18 mostra la sequenza di eventi e il diagramma di temporizzazione quando un dispositivo esterno con capacità di DMA richiede l'uso di un bus di sistema controllato al momento dall'MC68020. Innanzitutto, il dispositivo attiva (pone BASSA) la linea di segnale di richiesta di bus (*Bus Request:* $\overline{\text{BR}}$) per richiedere l'uso del bus. Quindi la CPU è costretta a rispondere con un segnale di concessione di bus (*Bus Grant:* $\overline{\text{BG}}$) all'inizio del ciclo di bus successivo. In risposta allo stato BASSO di $\overline{\text{BG}}$, il dispositivo attiva il segnale di riconoscimento della concessione di bus (*Bus Grant ACKnowledge:* $\overline{\text{BGACK}}$). Fintantoché la linea di segnale $\overline{\text{BGACK}}$ permane nello stato BASSO, il dispositivo richiedente agisce come master di bus temporaneo. Quando avrà completato le sue operazioni, tale dispositivo disattiverà la linea di segnale $\overline{\text{BGACK}}$ e l'MC68020 riprenderà il controllo come master di bus.

Nell'esempio rappresentato nella Fig. 13.18, l'interfaccia per il dispositivo DMA deve contenere la circuiteria per richiedere il bus ed assumere la gestione delle linee dei segnali di indirizzi, di dati e di controllo. Se più di un dispositivo può richiedere il bus contemporaneamente, la circuiteria esterna deve riconoscere le diverse richieste e determinare quale dispositivo sarà il prossimo master di bus. Il modulo di arbitrato di bus MC68452 della Motorola è un chip di circuiti integrati progettato per effettuare l'arbitrato fra otto dispositivi che possono agire da master di bus. Per ulteriori dettagli in merito all'arbitrato di bus, si rimanda il lettore ai vari riferimenti bibliografici relativi a questo capitolo, riportati nell'App. E alla fine del libro.

PROCESSORE

DISPOSITIVO RICHIEDENTE

Richiesta di bus

1) Attiva la richiesta di bus ($\overline{BR}$).

Concessione di arbitrato di bus

1) Attiva la concessione di bus ($\overline{BG}$).

Riconoscimento della padronanza del bus

1) L'arbitrato esterno determina il successivo master di bus
2) Il successivo master di bus resta in attesa della terminazione del ciclo corrente.
3) Il successivo master di bus attiva la linea di riconoscimento di concessione di bus ($\overline{BGACK}$) per divenire il nuovo master.
4) Il master di bus disattiva $\overline{BR}$.

Terminazione dell'arbitrato

1) Disattiva $\overline{BG}$ e attende la disattivazione di $\overline{BGACK}$.

Attività come master di bus

1) Esegue i trasferimenti di dati (cicli di lettura e di scrittura).

Rilascio della padronanza del bus

1) Disattiva $\overline{BGACK}$.

Riarbitrato o ripristino dell'attività del processore

CLK

A0-A31

FC0-FC2

SIZ0/SIZ1

R/$\overline{W}$

$\overline{ECS}$

$\overline{OCS}$

$\overline{AS}$

$\overline{DS}$

$\overline{DSACK0}$

$\overline{DSACK1}$

$\overline{DBEN}$

D0-D31

$\overline{BR}$

$\overline{BG}$

$\overline{BGACK}$

Processore → | ← Dispositivo DMA → | ← Processore

Fig. 13.18 *(a) Sequenza di arbitrato di bus. (b) Temporizzazione di arbitrato di bus. (Per gentile concessione di Motorola, Inc.)*

Il ciclo di lettura-modifica-scrittura. Un ciclo di lettura-modifica-scrittura ha luogo quando l'MC68020 esegue un'istruzione TAS (*Test And Set:* esamina e imposta), CAS (*Compare And Swap:* confronta e scambia), o CAS2 (*Compare And Swap 2:* confronta e scambia due volte).Durante l'esecuzione, esse realizzano il ciclo indivisibile, facendo sì che la CPU attivi (ponendo nello stato BASSO) la linea di segnale di lettura-modifica-scrittura ($\overline{RMC}$) per l'intera durata dell'operazione di lettura del valore in una locazione, modificando tale valore quando è necessario, e scrivendo il valore di nuovo al medesimo indirizzo.

Quando viene eseguita un'istruzione che causa un ciclo di lettura-modifica-scrittura, la CPU, che è il master di bus mentre esegue l'istruzione, non invierà un segnale di concessione del bus ($\overline{BG}$) in risposta ad un segnale di richiesta del bus ($\overline{BR}$).[10] Quindi, la CPU che esegue un'istruzione TAS, CAS o CAS2 non rilascerà il bus di sistema finché l'istruzione non sarà stata completata.

Errore di bus. La CPU inizia ad elaborare l'eccezione di errore di bus quando un dispositivo esterno attiva (ponendo nello stato BASSO) la linea di segnale di errore di bus ($\overline{BERR}$), per indicare che un ciclo di bus in corso di svolgimento non può essere completato. Il segnale di errore di bus è solitamente causato da un timer watchdog quando un dispositivo esterno non risponde durante un'operazione di trasferimento di dati. Durante un trasferimento, se il dispositivo non può attivare le proprie linee di segnali di riconoscimento del trasferimento di dati e della dimensione ($\overline{DSACK0}$, $\overline{DSACK1}$) in risposta ad un ciclo di lettura o di scittura della CPU, il timer watchdog dovrebbe attivare il segnale $\overline{BERR}$. Il segnale di errore di bus può essere utilizzato anche per indicare un tentativo di accesso ad una locazione in un'area protetta della memoria o un difetto di pagina in un sistema con memoria virtuale.[11]

Condizione di arresto del processore: il difetto di doppio bus. Se avviene un errore di indirizzo o un errore di bus durante l'elaborazione di eccezione per un errore d'indirizzo, un errore di bus o un'eccezione di reset, si dice che è avvenuto un difetto di *doppio bus.* La CPU riconosce questa condizione ed attiva (stato BASSO) la linea di segnale di arresto ($\overline{HALT}$), per indicare che non è in grado di continuare l'elaborazione. Soltanto un reset della CPU può far sì che il processore riprenda ad eseguire le istruzioni.

La linea di segnale RESET. Si tratta di una linea di segnale bidirezionale, utilizzata durante l'inizializzazione del sistema. Quando essa viene attivata dalla circuiteria esterna come ingresso all'MC68020, la CPU inizia l'elaborazione dell'eccezione di reset. Come segnale di uscita, la linea $\overline{RESET}$ è posta nello stato

[10] Se le linee di segnale $\overline{BERR}$ e $\overline{BR}$ vengono poste simultaneamente nello stato BASSO durante un ciclo RMC, la CPU rilascerà il bus e ritenterà il ciclo RMC quando tali segnali saranno negati.

[11] Altri impieghi della linea di segnale di errore di bus ($\overline{BERR}$) sono stati descritti nel par. 11.5. In alcuni casi, il segnale di errore di bus farà sì che la CPU avvii l'elaborazione di un'eccezione diversa dall'errore di bus.

BASSO dalla CPU quando viene eseguita l'istruzione RESET. Questa istruzione serve a far sì che i dispositivi esterni inizializzino la loro circuiteria d'interfaccia.[12] L'eccezione di reset per la CPU è stata descritta nel cap. 10. L'istruzione RESET è stata discussa nei parr. 10.2 e 13.2.

13.4.6 L'interfaccia di coprocessore

Un diagramma semplificato dell'interfaccia di coprocessore è mostrato nella Fig. 13.19. Le linee di segnale del codice di funzione {111} e le linee dei segnale d'indirizzo da A13 a A19 selezionano il coprocessore quando la CPU esegue un'istruzione di linea F come descritto nel par. 12.2. L'accesso avviene nello spazio di CPU, come descritto nel sottopar. 13.4.3. Il trasferimento di comandi e di dati tra il coprocessore e la CPU avviene mediante i normali cicli di trasferimento di I/O descritti nel sottopar. 13.4.1.

FC2-FC0 = 111 → Ciclo di spazio di CPU
A19-A16 = 0010 → Accesso di coprocessore nello spazio di CPU
A15-A13 = xxx → Identificazione del coprocessore
A4-A1 = rrrr → Selettore del registro d'interfaccia del coprocessore

*La logica di selezione del chip può essere integrata nel coprocessore

Le linee d'indirizzo non specificate sono "0" durante l'accesso del coprocessore.

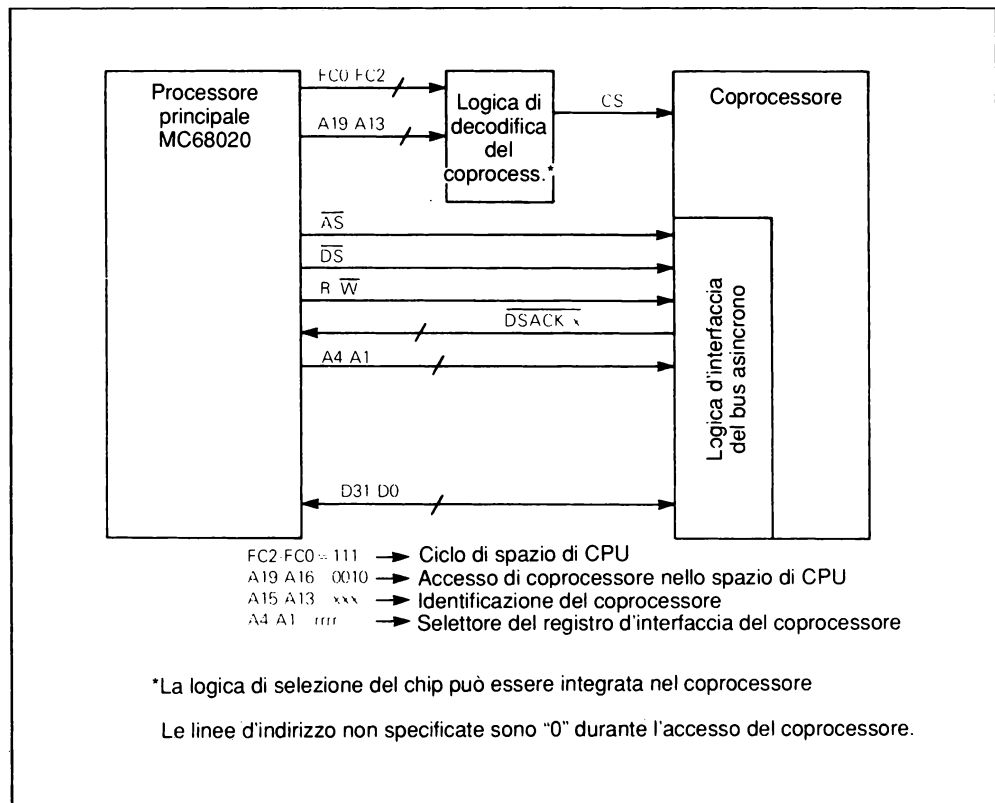



Fig. 13.19 *Interfaccia di coprocessore. (Per gentile concessione di Motorola, Inc.)*

[12] La Motorola consiglia di mantenere BASSO il segnale $\overline{RESET}$ per 520 cicli di clock quando viene inviato alla CPU. L'istruzione RESET fa sì che la linea di segnale $\overline{RESET}$ resti BASSA per 512 cicli di clock.

ESERCIZI

13.4.1

Viene eseguita l'istruzione

```
MOVE.B    D1,$2001
```

Si definiscano i valori o le condizioni per le linee d'indirizzo, le linee di dati e le altre linee di segnale quando l'operando è trasferito nella memoria.

13.4.2

Si determini il tipo di riferimento alla memoria ed i valori delle linee del codice di funzione per ciascuna della seguenti istruzioni:

```
(a) MOVE.W   D1,(A1)        ; IN MODO UTENTE
(b) JSR      (A1)           ; IN MODO UTENTE
(c) MOVE.W   D1,DISP(PC)    ; IN MODO SUPERVISORE
(d) CLR.L    $2000          ; IN MODO SUPERVISORE
```

13.4.3

Quale indirizzo di vettore nella tabella dei vettori di eccezione viene richiesto da un dispositivo che ha fornito il numero di vettore 64 in risposta ad un'interruzione? Potrebbe un'interfaccia fornire numeri di vettori compresi tra 2 e 47 e, se sì, quali problemi potrebbero sorgere nel sistema?

13.4.4

Si definisca il comportamento del sistema se le linee di richiesta d'interruzione ricevono i seguenti livelli:

$\overline{IPL2}$ è BASSO
$\overline{IPL1}$ è ALTO
$\overline{IPL0}$ è ALTO

Si definisca la sequenza di operazioni ed i valori delle linee d'indirizzo e delle linee del codice di funzione.

13.4.5

Si disegni il diagramma di temporizzazione per un trasferimento di longword ad una porta di 8 bit.

13.4.6

Si disegni il diagramma di temporizzazione per un trasferimento di longword ad una porta di 32 bit usando l'istruzione

```
MOVE.L    D1,$10003
```

come esempio.

13.4.7

Si consideri un sistema con memoria virtuale con software di supervisore, in cui lo spazio di programma e lo spazio di dati iniziano al medesimo indirizzo virtuale. Dopo che si è verificato un difetto di pagina durante l'esecuzione del programma, in che modo la routine di gestione del difetto di pagina può trasferire le nuove istruzioni nell'area di memoria nello spazio di programma di supervisore, considerato che tutte le scritture nella memoria sono accessi allo spazio di dati? In che modo tale routine può essere modificata dalla presenza di un controllore di DMA, impiegato per trasferire le informazioni tra un'unità di memoria a disco e la memoria centrale?

CAPITOLO 14

IL VMEbus ED I RELATIVI SISTEMI

14.0 INTRODUZIONE

In questo capitolo saranno trattati essenzialmente il VMEbus ed i computer che utilizzano questo sistema di bus. Le caratteristiche fisiche ed elettriche del VMEbus saranno spiegate nei primi due paragrafi del capitolo, dopo che i bus di microcomputer saranno stati descritti in questa introduzione. I componenti del VMEbus ed i sistemi completi di computer che utilizzano il VMEbus saranno descritti nei parr. 14.3 e 14.4, rispettivamente. Il par. 14.5 presenterà le tecniche per la selezione e l'avviamento del sistema. Nell par. 14.6 finale saranno discussi altri bus di sistemi di microcomputer ed i computer basati sull'MC68020 che impiegano tali bus.

Bus di microcomputer. La funzione del bus di sistema di un microcomputer è quella di trasferire le informazioni nella forma di segnali elettrici tra i diversi moduli di un sistema di computer. Il modulo tipico è una singola piastra a circuito stampato o "scheda", come quella dei moduli VME presentati nel par. 1.2. I moduli hanno connettori che ne consentono l'inserimento in connettori corrispondenti montati sulla piastra-madre (*backplane*) del sistema del computer. Questa è una piastra a circuito stampato su cui sono realizzati i connettori di accoppiamento e le linee di segnale parallele che rappresentano il bus. La struttura funzionale di questo tipo di sistema modulare è mostrata nella Fig. 14.1. I componenti all'interno del rettangolo trattaggiato nella figura rappresentano il bus di sistema e la relativa circuiteria d'interfacciamento per la connessione al bus dei chip a circuiti integrati del modulo.

Fisicamente la piastra-madre fa parte di una struttura che contiene un alimentatore e gli alloggiamenti (*slot*) per i moduli da inserire. Tale struttura ha varie denominazioni: telaio (*chassis*), gabbia di schede, o *subrack*. Per i sistemi VMEbus, le dimensioni ed altre caratteristiche meccaniche del subrack che contiene i

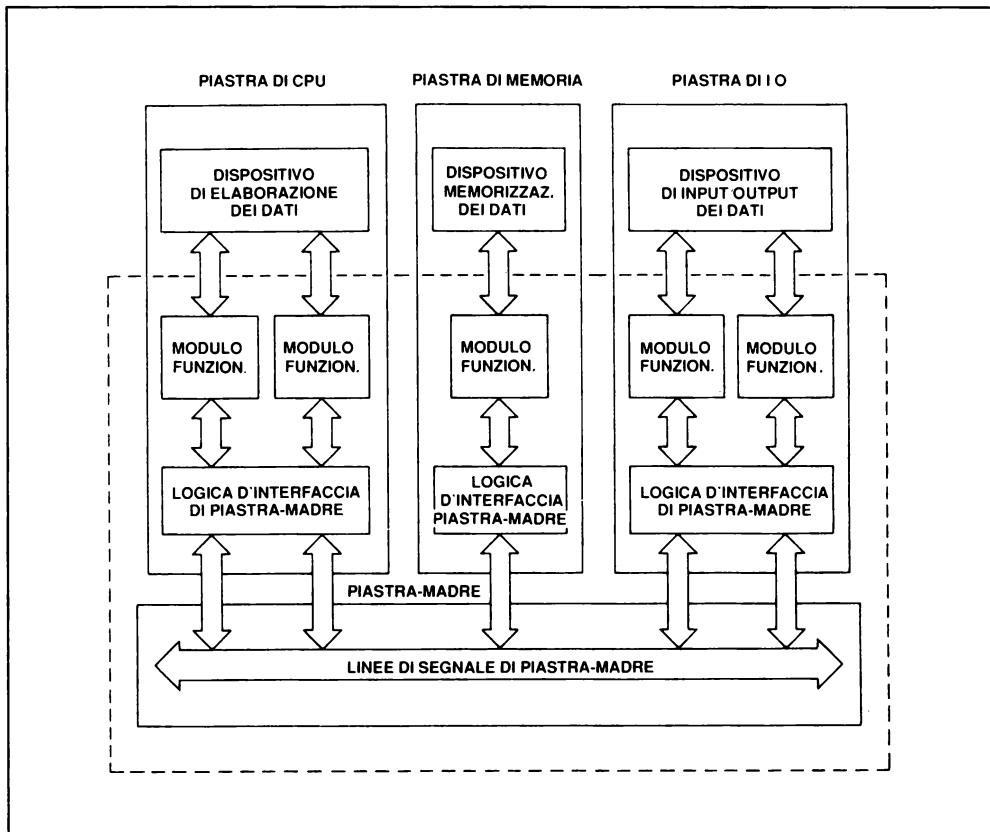



Fig. 14.1 *Struttura funzionale di un bus di microcomputer e moduli di computer.*

moduli VME sono definite esattamente nelle specifiche del VMEbus, come sarà descritto nel par. 14.1. Quando sia le caratteristiche meccaniche che quelle elettriche di un bus di microcomputer — come il VMEbus — sono definite esattamente ed ampiamente conosciute, esso è denominato *bus standard.*[1]

I concetti di utilizzazione di un bus e di costruzione modulare di un sistema di computer hanno come scopo primario la riduzione del numero di interconnessioni tra gli elementi circuitali. Un sistema siffatto permette anche di ottenere una certa flessibilità nel progetto del sistema, poiché si possono facilmente aggiungere dei moduli al sistema dopo che è stata realizzata la configurazione di base. Ogni nuovo modulo deve semplicemente soddisfare le specifiche del bus per essere compatibile col sistema esistente. Quando s'impiega un bus standard, produttori indipendenti possono produrre i propri moduli di computer che vanno inseriti nel bus standard. Nel caso dei moduli VMEbus, dozzine di produttori producono

[1] Esistono bus di microcomputer definiti di *proprietà riservata*. In genere, tali bus sono utilizzati dal produttore in prodotti che non possono essere modificati né ampliati da altri produttori.

moduli che sono compatibili col VMEbus.[2] Ciò consente al progettista di un sistema di computer modulare di selezionare moduli da vari produttori per soddisfare i requisiti del sistema.

Molti bus di computer sono progettati tenendo presente un processore specifico. Per esempio, il VMEbus è stato progettato dalla Motorola e da altre società per la linea di processori dell'MC68020. Quindi, le linee di segnale del VMEbus sono simili nelle denominazioni e nelle funzioni alle linee di segnale delle CPU MC68000 e MC68020 della Motorola. Tuttavia, i segnali del VMEbus non sono identici a quelli del bus di CPU descritti nei parr. 2.1 e 13.4. Per motivi di chiarezza, il VMEbus in questo capitolo sarà denotato come *bus di sistema.* Il bus della CPU o il bus interno contenuto su moduli con un processore MC68020 sarà denominato *bus locale*. Questa distinzione è mostrata nella Fig. 14.2. Il bus locale è usato per trasferire le informazioni tra i componenti del modulo della CPU. Le comunicazioni tra moduli separati avvengono mediante il VMEbus. Per esempio, il modulo MVME133, già descritto nel par. 13.2, contiene sia un bus locale che i connettori per il VMEbus. Questo modulo sarà discusso in maggior dettaglio nel par. 14.4.

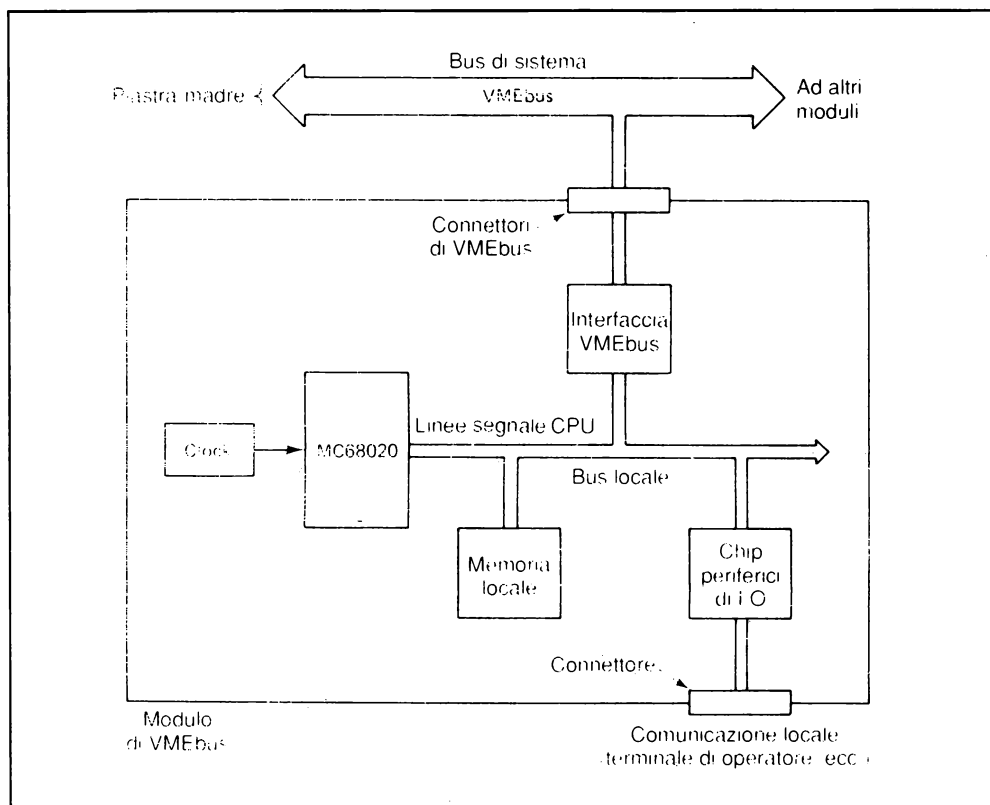



Fig. 14.2 *Modulo di CPU con un bus locale e connessioni di VMEbus.*

[2] Nel momento in cui si scrive, esistono oltre 100 società produttrici di moduli VMEbus.

Sebbene la compatibilità hardware sia posta in evidenza in molte trattazioni di sistemi di computer standard, lo scopo ultimo di un sistema di computer è l'esecuzione di programmi che servono a qualche applicazione utile. Per un sistema VMEbus, dev'essere selezionato un sistema operativo che tragga vantaggio dalle capacità del bus. Il software dovrebbe includere routine di eccezione standard e speciali, se necessario. Tali routine servono a controllare vari moduli che fanno parte del sistema di computer. Varie considerazioni di software per i sistemi VMEbus saranno discusse ulteriormente nei parr. 14.3. e 14.4.

14.1 CARATTERISTICHE GENERALI DEL VMEbus

In accordo col documento *VMEbus Specification*, il VMEbus si è sviluppato dal VERSAbus della Motorola. Questo era impiegato come piastra-madre di un sistema di sviluppo basato sull'MC68000, denominato EXORmacs. Nel 1981, il gruppo di microsistemi europeo della Motorola sviluppò il bus "Versa Module Europe", ora noto come VMEbus. Questo nuovo bus differiva dal VERSAbus in quanto consentiva l'impiego di moduli progettati su una "Eurocard", che era fisicamente più piccola delle piastre a circuito stampato del VERSAbus. Un certo numero di società negli Stati Uniti e in Europa accettò il nuovo bus ed esso divenne allora uno standard non ufficiale per le società produttrici di moduli per sistemi basati sull'MC68000 che utilizzavano il VMEbus. Lo standard divenne ufficiale allorché il VMEbus fu riconosciuto ed approvato dall'IEEE (*Institute of Electrical and Electronic Engineers*). Tale standard è noto come IEEE P1014 negli Stati Uniti e come IEC 821 (*International Eurotechnical Commission*) in Europa. La *VMEbus Specification*, elencata nei riferimenti bibliografici relativi a questo capitolo nell'App. E, è lunga 251 pagine. Essa fornisce le specifiche fisiche ed elettriche per il VMEbus, con dettagli sufficienti a consentire la produzione di moduli VMEbus da parte di qualsiasi azienda che disponga delle necessarie capacità tecniche e produttive.[3]

Di seguito sono elencati tre dei principali obbiettivi delle specifiche del VMEbus.

(a) Consentire la comunicazione tra dispositivi sul VMEbus senza disturbare le attività interne di altri dispositivi interfacciati al VMEbus.

(b) Specificare le caratteristiche elettriche e meccaniche richieste per progettare dispositivi che comunicheranno affidabilmente e senza ambiguità con altri dispositivi interfacciati al VMEbus.

(c) Specificare protocolli che definiscano in modo preciso l'interazione tra il VMEbus e i dispositivi interfacciati ad esso.

In breve, i moduli di CPU, i moduli di memoria, e i dispositivi periferici controllati dai moduli di I/O possono essere interconnessi tramite il VMEbus, se le specifiche sono rispettate. I dettagli di tali specifiche includono dati meccanici come

[3] Nella pratica, i moduli cosiddetti "VMEbus-compatibili" non sempre sono completamente compatibili. Si faccia attenzione! Vari riferimenti bibliografici relativi a questo capitolo, riportati nell'App. E alla fine del libro, discutono i problemi di incompatibilità.

quelli descritti in questo paragrafo. Le caratteristiche elettriche e funzionali del VMEbus saranno descritte nel par. 14.2.

14.1.1 Caratteristiche fisiche dei sistemi del VMEbus

La Fig. 14.3 mostra un telaio tipico, denominato *subrack* nella terminologia del VMEbus. Tale struttura contiene sia il VMEbus che la sua piastra-madre ed un alimentatore. Ha una larghezza di 19 pollici (circa 48 cm) per adattarsi al rack elettrico standard che può contenere uno o più di tali subrack. Ogni *slot* nel subrack è un alloggiamento in cui si può inserire una piastra per la connessione alla piastra-madre del VMEbus. Sono consentiti fino a 21 slot dalla specifica del VMEbus. Molti subrack permettono due dimensioni dei moduli VMEbus.

I moduli VMEbus sono costruiti su piastre a circuito stampato o su schede che devono rispettare esattamente le specifiche di altezza e lunghezza. Una piastra ad

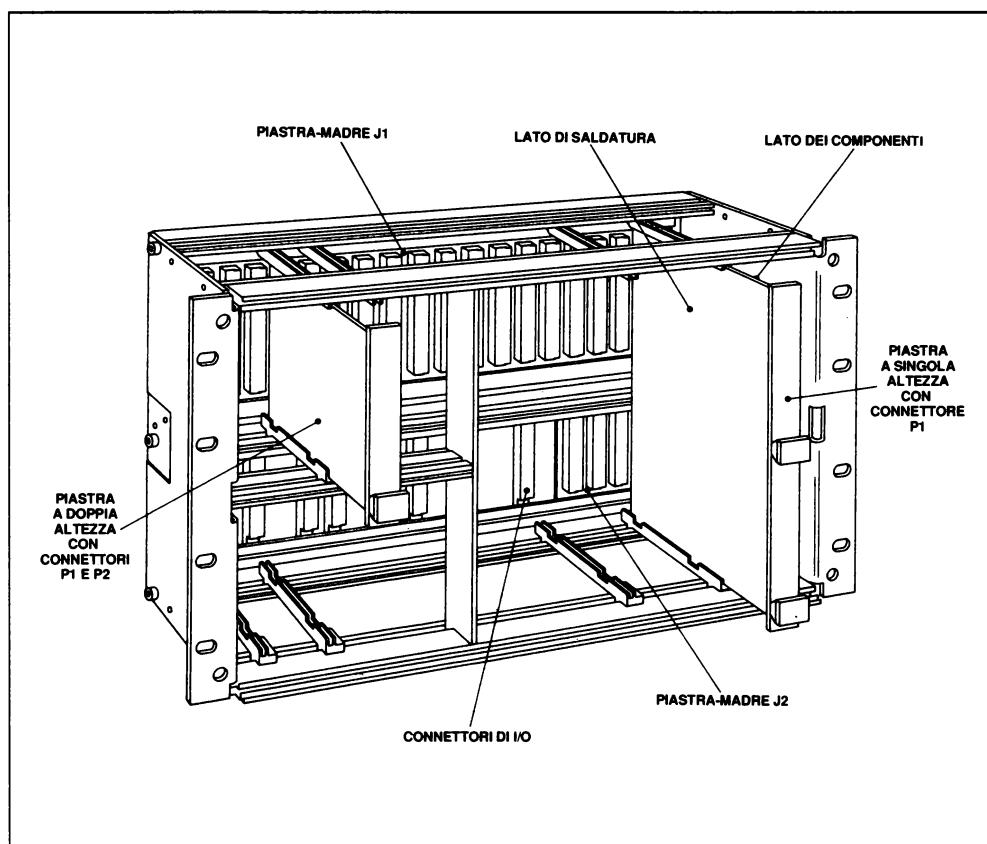



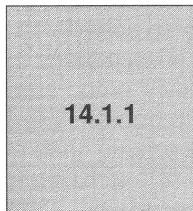
Fig. 14.3 *Esempio di subrack di VMEbus. (Per gentile concessione di Motorola, Inc.)*

altezza singola è alta 100 mm e lunga 160 mm.[4] Essa occupa una metà dell'altezza di uno slot di piastra, come mostrato nella Fig. 14.3. La piastra a doppia altezza è impiegata per il modulo MVME133 illustrato nella Fig. 1.1.

14.1.2 Connettori del VMEbus

Ogni piastra del VMEbus ha almeno un connettore a 96 piedini, designato come "P1", che s'inserisce in un connettore corrispondente designato come "J1" sulla piastra-madre. La piastre a doppia altezza hanno solitamente due connettori, P1 e P2. Ciascun connettore ha tre file di 32 piedini. I segnali che corrispondono alle designazioni dei piedini saranno presentati nel par. 14.2.

ESERCIZIO

14.1.1

Lo standard Eurocard definisce piastre a singola, doppia, tripla e quadrupla altezza. Esistono anche piastre "estese in profondità", di lunghezza superiore a 160 mm. Si discutano i vantaggi e gli svantaggi di piastre di maggiori dimensioni per i sistemi VMEbus.

14.2 OPERAZIONI DEL VMEbus, CANALI DI I/O ED IL VSBbus

Questo paragrafo riguarda innanzitutto l'aspetto funzionale della circuiteria d'interfaccia e le linee di segnale sul VMEbus. Queste linee di segnale consentono di effettuare il trasferimento di dati, le interruzioni, l'arbitrato di bus e di svolgere altre funzioni, come richiesto dai moduli del VMEbus. Quindi il VMEbus agisce da bus di sistema, connettendo i moduli e permettendo ad essi di comunicare tra loro. Nella maggior parte dei sistemi, viene adottato questo progetto di bus singolo. Tuttavia, in alcune applicazioni, è desiderabile avere ulteriori bus o percorsi alternativi per il trasferimento di dati quando il VMEbus è inadeguato. La Motorola ha progettato vari suoi subrack per contenere bus indipendenti che potenzino il VMEbus e che soddisfino i requisiti speciali di certe applicazioni. L'*I/O Channel* (canale di I/O) è un'estensione di un bus locale di modulo di CPU, che provvede alle funzioni d'ingresso/uscita per varie applicazioni di tipo speciale. Un altro bus, designato come VSBbus (*VME subsystem bus:* bus di sottosistema VME), può essere incluso in un sistema per fornire le capacità di trasferimento di dati ad alta velocità e di comunicazione tra moduli indipendenti del VMEbus. In questo paragrafo saranno descritti il VMEbus, l'I/O channel e il VSBbus.[5]

---

[4] La dimensione di una piastra ad altezza singola del VMEbus è quella di una piastra a circuito stampato "Eurocard".

[5] Un sottoinsieme del VMEbus, denotato come VMSbus, può essere usato per la comunicazione seriale tra i moduli. Questo bus è descritto in un articolo nei riferimenti bibliografici relativi a questo capitolo, elencati nell'app. E alla fine del libro.

Notazioni per le linee di segnale. Le linee di segnale dell'MC68020 sono state definite in dettaglio nel par. 13.4. In particolare, il loro stato di attività è stato descritto in termini dei valori ALTO e BASSO di tensione sulla linea di segnale. I segnali "veri" nello stato BASSO sono stati designati da una sbarretta di sopralineatura [per esempio, $\overline{AS}$ (*Address Strobe:* abilitazione dell'indirizzo) è "vera", cioè attiva, nella condizione di tensione BASSA]. Le linee di segnale del VMEbus sono designate diversamente quando sono nello stato BASSO, in accordo con la *VMEbus Specification*. In tale documento, il nome mnemonico di un segnale attivo BASSO è seguito da un asterisco (*). Quindi, il segnale di abilitazione d'indirizzo del VMEbus è designato come AS*. In questo paragrafo si adotterà la convenzione del par. 13.4 per designare le linee di segnale della CPU, ma lo stato attivo BASSO di una linea di segnale del VMEbus sarà denotato da un asterisco dopo il nome.

14.2.1 Le operazioni del VMEbus

La Fig. 14.4 illustra un sistema VMEbus di esempio nella forma di diagramma a blocchi. I componenti al di sotto della linea tratteggiata che corre per tutta la lunghezza del bus nella figura sono i circuiti richiesti per un'interfaccia del VMEbus. Un *controllore di sistema* è mostrato come una piastra separata nella Fig. 14.4, benché esso di solito faccia parte della piastra di CPU del computer. Il controllore di sistema fornisce i segnali di temporizzazione, l'arbitrato di bus ed il controllo della priorità d'interruzione per il VMEbus. Ogni sistema contiene la circuiteria del controllore di sistema che deve risiedere nello slot 1 del subrack del VMEbus. Gli altri moduli sono selezionati o progettati ed inseriti nel subrack in conformità coi requisiti di una particolare applicazione. Quando il sistema è realizzato da moduli VMEbus, il produttore del modulo fornirà i circuiti d'interfaccia per connettere il modulo al VMEbus. Il progetto di tale circuiteria è discusso in vari riferimenti bibliografici relativi a questo capitolo, elencati nell'App. E alla fine del libro.

14.2.2 Le linee di segnale del VMEbus

La Fig. 14.4 mostra le linee di segnale del VMEbus suddivise in quattro gruppi distinti, che sono definiti e descritti brevemente nella Tab. 14.1. La Tab. 14.2 mostra le assegnazioni dei piedini per i connettori del VMEbus. In un certo istante, un solo dispositivo svolge la funzione di *master di bus*. Esso può controllare il bus ed effettuare i trasferimenti di dati utilizzando le linee di segnale del bus di trasferimento dei dati (*Data Transfer Bus:* DTB) e del bus di arbitrato.[6] Un dispositivo *slave* ("schiavo") deve rispondere correttamente o inviare una richiesta di errore di bus in maniera simile a quella dei trasferimenti dalla CPU al dispositivo, descritti nel par. 13.4. Le transazioni di bus sono sincronizzate da segnali di temporizzazione generati dal clock di sistema. Le informazioni di temporizzazione ed il protocollo specifico per definire le interazioni di vari segnali di bus non saranno discusse qui poiché sono descritte nel documento *VMEbus Specification*.

[6] Questi sono sottoimpieghi del VMEbus. La terminologia adottata in questo paragrafo è coerente con quella della *VMEbus Specification*.

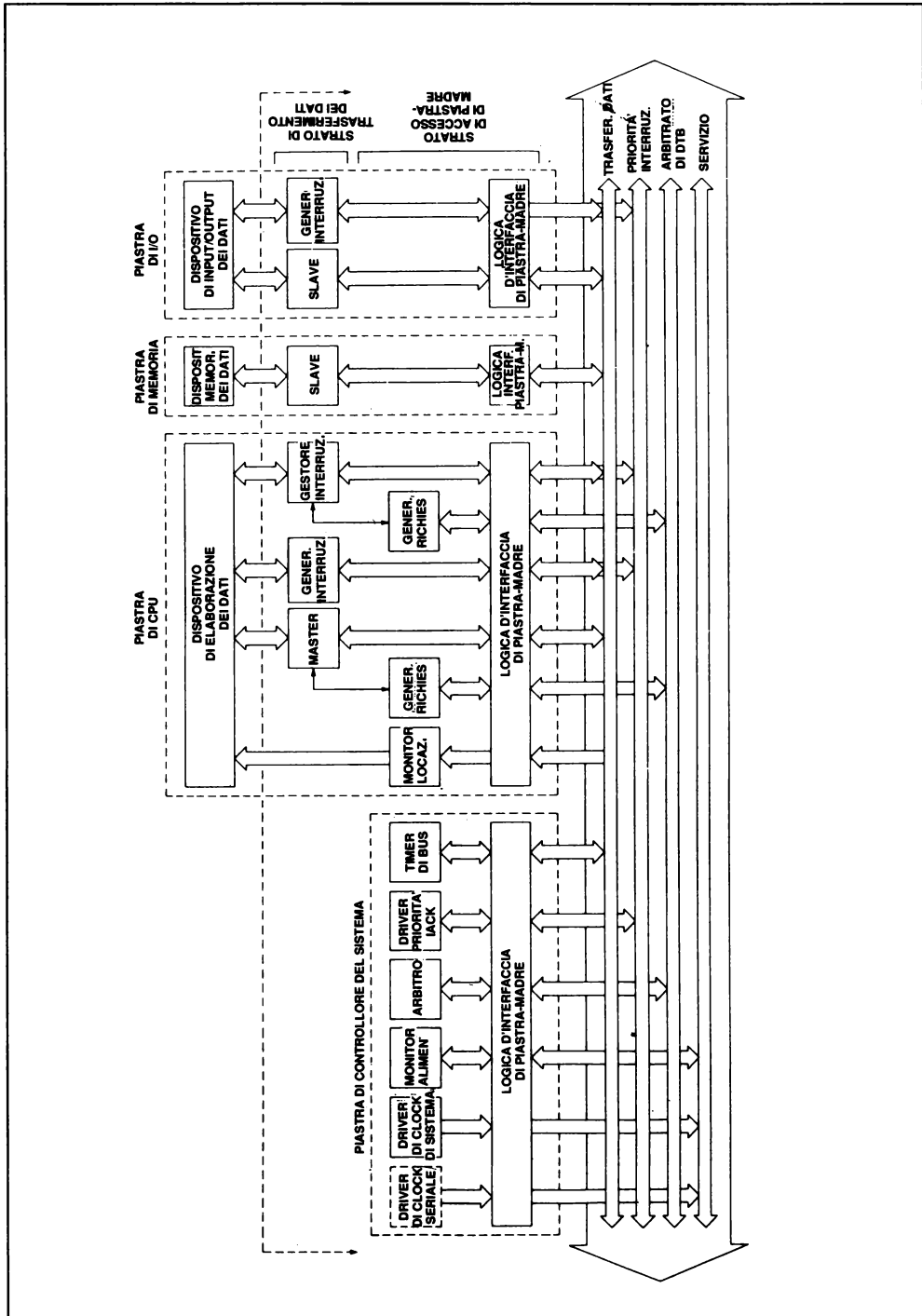



Fig. 14.4 *Diagramma a blocchi funzionale del sistema VMEbus. (Per gentile concessione di Motorola, Inc.)*

Tab. 14.1 *Gruppi di linee di segnale di VMEbus.*

Nome del sottobus	Linee di segnale	Scopo
Bus di trasferimento dati (*Data Transfer Bus:* DTB)	Linee d'indirizzo, linee di dati e linee di controllo	Trasferimento di dati tra dispositivi master e slave.
Arbitrato di bus di trasferimento dati (*Data Transfer Bus Arbitration*)	Quattro linee di richiesta di bus, linee di concessione di bus e linee di bus occupato.	Arbitrato tra richieste dai master di bus.
Bus d'interruzione di priorità (*Priority Interrupt*)	Sette linee di richiesta d'interruzione e linee di riconoscimento d'interruzione	Richieste d'interruzione.
Bus di servizio (*Utility Bus*)	Clock, reset, guasto AC, guasto di sistema, e linee di dati seriali	Temporizzazione periodica, inizializzazione e scopi diagnostici.

Il circuito di arbitrato della Fig. 14.4 è impiegato per accettare e riconoscere le richieste su uno di sette livelli di priorità selezionati sulle linee dei segnali d'interruzione. Questi segnali sono simili a quelli per le interruzioni dell'MC68020 definiti nel par. 13.4. Quando un'interruzione viene riconosciuta dalla circuiteria di gestione dell'interruzione (DRIVER PRIORITA' IACK in Fig. 14.4), il bus di trasferimento di dati ed il bus di arbitrato sono impiegati per consentire al dispositivo interrompente d'interrompere la CPU. Nei sistemi basati sull'MC68020, il dispositivo in questione e la circuiteria di gestione delle interruzioni deve seguire il protocollo definito nel par. 13.4 per far sì che la CPU riconosca l'interruzione ed inizi ad elaborare l'interruzione.

Le linee di segnale del bus di servizio includono un clock a 16 Mhz che è situato nel modulo del controllore di sistema. Questo clock serve a definire i cicli di bus per i trasferimenti di dati ed altre operazioni sul VMEbus.[7] Vari altri segnali sul bus di servizio sono usati per indicare malfunzionamenti nel sistema. Un'indicazione di questo tipo causerebbe l'avviamento di una sequenza di errore di bus da parte della CPU oppure causerebbe un'interruzione ad alta priorità verso la CPU. Qualunque azione intrapresa dipende interamente dalla routine di gestione dell'eccezione che viene eseguita allorché il malfunzionamento è stato riconosciuto.

[7] Rispetto alla temporizzazione della CPU, i trasferimenti sono asincroni, come descritto nel par. 13.4.

La Tab. 14.2 elenca i segnali connessi al connettore P1, tra cui 23 linee di segnali d'indirizzo e 16 linee di segnali di dati.[8] Ciò corrisponde alla capacità d'indirizzamento di 24 bit ed al bus di dati di 16 bit della CPU MC68000. Nei sistemi basati sull'MC68020 che impiegano l'intera capacità del processore di 32 bit per l'indirizzo e di 32 bit per i trasferimenti, la seconda riga (RIGA b) di P2 dev'essere connessa. I 64 segnali di P2 designati come "definiti dall'utente" possono essere usati per qualsiasi scopo. Essi non sono connessi al VMEbus.

Tab. 14.2 *Assegnazioni dei piedini dei connettori di VMEbus.* *(a)* P1/J1.

Numero del piedino	Nome mnemonico del segnale della RIGA a	Nome mnemonico del segnale della RIGA b	Nome mnemonico del segnale della RIGA c
1	D00	BBSY*	D08
2	D01	BCLR*	D09
3	D02	ACFAIL*	D10
4	D03	BG0IN*	D11
5	D04	BG0OUT*	D12
6	D05	BG1IN*	D13
7	D06	BG1OUT*	D14
8	D07	BG2IN*	D15
9	GND	BG2OUT*	GND
10	SYSCLK	BG3IN*	SYSFAIL*
11	GND	BG3OUT*	BERR*
12	DS1*	BR0*	SYSRESET*
13	DS0*	BR1*	LWORD*
14	WRITE*	BR2*	AM25
15	GND	BR3*	A23
16	DTACK*	AM0	A22
17	GND	AM1	A21
18	AS*	AM2	A20
19	GND	AM3	A19
20	IACK*	GND	A18
21	IACKIN*	SERCLK(1)	A17
22	IACKOUT*	SERDAT*(1)	A16
23	AM4	GND	A15
24	A07	IRQ7*	A14
25	A06	IRQ6*	A13
26	A05	IRQ5*	A12
27	A04	IRQ4*	A11
28	A03	IRQ3*	A10
29	A02	IRQ2*	A09
30	A01	IRQ1*	A08
31	-12V	+5VSTDBY	+12V
32	+5V	+5V	+5V

[8] I segnali di abilitazione di dati (DS0* e DS1*) indicano lo stato della linea di segnale d'indirizzo A0 della CPU. Quindi l'intervallo d'indirizzamento completo è di 2^{32} byte, corrispondente a 24 linee di segnali d'indirizzo.

Tab. 14.2 *(continuazione)* *(b)* P2/J2.

Numero del piedino	Nome mnemonico del segnale della RIGA a	Nome mnemonico del segnale della RIGA b	Nome mnemonico del segnale della RIGA c
1	Definito dall'utente	+5V	Definito dall'utente
2	Definito dall'utente	GND	Definito dall'utente
3	Definito dall'utente	Riservato	Definito dall'utente
4	Definito dall'utente	A24	Definito dall'utente
5	Definito dall'utente	A25	Definito dall'utente
6	Definito dall'utente	A26	Definito dall'utente
7	Definito dall'utente	A27	Definito dall'utente
8	Definito dall'utente	A28	Definito dall'utente
9	Definito dall'utente	A29	Definito dall'utente
10	Definito dall'utente	A30	Definito dall'utente
11	Definito dall'utente	A31	Definito dall'utente
12	Definito dall'utente	GND	Definito dall'utente
13	Definito dall'utente	+5V	Definito dall'utente
14	Definito dall'utente	D16	Definito dall'utente
15	Definito dall'utente	D17	Definito dall'utente
16	Definito dall'utente	D18	Definito dall'utente
17	Definito dall'utente	D19	Definito dall'utente
18	Definito dall'utente	D20	Definito dall'utente
19	Definito dall'utente	D21	Definito dall'utente
20	Definito dall'utente	D22	Definito dall'utente
21	Definito dall'utente	D23	Definito dall'utente
22	Definito dall'utente	GND	Definito dall'utente
23	Definito dall'utente	D24	Definito dall'utente
24	Definito dall'utente	D25	Definito dall'utente
25	Definito dall'utente	D26	Definito dall'utente
26	Definito dall'utente	D27	Definito dall'utente
27	Definito dall'utente	D28	Definito dall'utente
28	Definito dall'utente	D29	Definito dall'utente
29	Definito dall'utente	D30	Definito dall'utente
30	Definito dall'utente	D31	Definito dall'utente
31	Definito dall'utente	GND	Definito dall'utente
32	Definito dall'utente	+5V	Definito dall'utente

Nota: Un segnale seguito da un asterisco è vero (attivo) quando è BASSO.
Fonte: Motorola, Inc.

14.2.3 Il VMEbus e le linee di segnale dell'MC68020

Poiché il VMEbus è un bus di sistema di finalità generale, le linee di segnale del VMEbus non corrispondono esattamente alle linee di segnale dell'MC68020, secondo la definizione del par. 13.4. Presupponendo che l'MC68020 sia il processore del modulo di CPU, la circuiteria d'interfacciamento del modulo deve fornire la temporizzazione e il protocollo per far sì che i segnali della CPU corrispondano

ai segnali del VMEbus. L'incompatibilità fondamentale è mostrata nella Tab. 14.3, in cui i segnali del VMEbus sono classificati in base ai sottobus descritti in precedenza. Nella maggior parte dei sistemi basati sull'MC68020, soltanto le linee dei segnali d'indirizzo, di dati, R/$\overline{W}$, $\overline{AS}$, $\overline{BERR}$ e $\overline{RESET}$ corrispondono direttamente

Tab. 14.3 *Segnali del VMEbus e dell'MC68020.*

VMEbus	Segnale dell'MC68020	Nome in VME
Bus di trasferimento dati		
A00-A31	A00-A31	Indirizzo
D00-D31	D00-D31	Dati
WRITE*	R/$\overline{W}$	Scrittura
AS*	$\overline{AS}$	Abilitazione indirizzo
DS0* DS1*	—	Abilitazione dati
LWORD*	—	Longword
DTACK*	—	Riconoscimento del trasferimento di dati
BERR*	$\overline{BERR}$	Errore di bus
AM0-AM5	—	Modificatore d'indirizzo
Bus di arbitrato di DTB		
BR0*-BR3*	$\overline{BR}$ (una linea)	Richiesta di bus (quattro linee)
BG0OUT* – BG3OUT*	—	Uscita concessione bus
BBSY*	—	Bus occupato
BCLR*	—	Bus libero
BG0IN*-BG3IN*	—	Ingresso concessione bus
Bus d'interruzione di priorità		
IRQ1*-IRQ7*	$\overline{IPL0}$-$\overline{IPL2}$ (3 linee)	Richiesta d'interruzione (sette linee)
IACK*, IACKIN*, IACKOUT*	—	Riconoscimento dell'interruzione
Bus di servizio		
SYSCLK	CLK	Clock di sistema
SERCLK	—	Clock seriale
SERDAT*	—	Dati seriali
ACFAIL*	—	Guasto AC
SYSRESET*	$\overline{RESET}$	Reset di sistema
SYSFAIL*	—	Guasto di sistema

ai segnali del VMEbus. I segnali di clock (SYSCLK e CLK) non sono necessariamente gli stessi.

Quando l'MC68020 diviene il master di bus, i suoi altri segnali devono essere convertiti in segnali di VMEbus per effettuare i trasferimenti di dati o svolgere altre operazioni, come l'indirizzamento dello spazio di CPU per riconoscere un'interruzione o per comunicare con un coprocessore. In un'operazione di trasferimento di dati, per esempio, la CPU attiva i segnali del codice di funzione, i segnali di dimensione ed altri segnali per avviare il trasferimento. Essa attende una risposta dei segnali $\overline{DSAKC0}$ e $\overline{DSACK1}$ (*Data transfer and Size ACKnowledge:* riconoscimento del trasferimento di dati e della dimensione) per completare il trasferimento. Chiaramente, certe linee di segnale del VMEbus devono essere assegnate dalla CPU e da un dispositivo esterno per effettuare i trasferimenti di dati ed operazioni simili. I segnali specifici utilizzati dipendono dal progetto della circuiteria d'interfaccia e non sono standard. L'attività della CPU e le linee di segnale, tuttavia, sono definite esattamente. Si consiglia al lettore di rivedere il par. 13.4, in cui è stata discussa ciascuna linea di segnale della CPU.

14.2.4 Il canale di I/O (*I/O Channel*)

L'*I/O Channel* (canale di I/O) definito dalla Motorola permette di connettere dispositivi periferici o moduli speciali ad un bus indipendente dal VMEbus. Tale configurazione è mostrata nella Fig. 14.5. Si nota un modulo da VMEbus ad I/O Channel (MVME316) che consente la comunicazione tra il bus dell'I/O Channel ed il VMEbus. Lo scopo è quello di rendere possibili i trasferimenti di dati ed altre operazioni sul canale di I/O, senza disturbare l'attività sul VMEbus. La comunicazione tra i due bus è impiegata per inizializzare il modulo del canale di I/O o per trasferire i dati alla memoria quando il modulo del canale di I/O ha completato le sue operazioni. Normalmente i moduli di I/O Channel sono caratterizzati da velocità di trasferimento molto basse rispetto a quelle del VMEbus.[9]

Il modulo d'interfaccia denotato con la sigla MVME316 nella Fig. 14.5 è inseribile in uno slot del VMEbus. Gli altri moduli sono realizzati su piastre di VMEbus ad altezza singola. Un bus di I/O Channel ha una capacità d'indirizzamento di 12 bit ed un bus di dati di 8 bit con quattro livelli d'interruzione. La connessione del canale può utilizzare i piedini del connettore P2 del VMEbus che non sono usati dal VMEbus, come definito nella Tab. 14.2. Le piastre di I/O Channel possono essere ospitate in uno slot ad altezza singola di un subrack del VMEbus, o possono essere connesse al sistea VMEbus mediante un cavo flessibile lungo fino a 5 metri. Un tipico modulo di I/O Channel può servire per l'acquizione di ingressi analogici, per l'I/O seriale, o per emettere segnali analogici.

[9] La velocità di trasferimento tipica è di 2 megabyte/secondo sull'I/O Channel, contro i 40 megabyte/secondo del VMEbus. Comunque, le velocità di trasferimento effettive dipendono dallo specifico progetto del VMEbus e del modulo dell'I/O Channel.

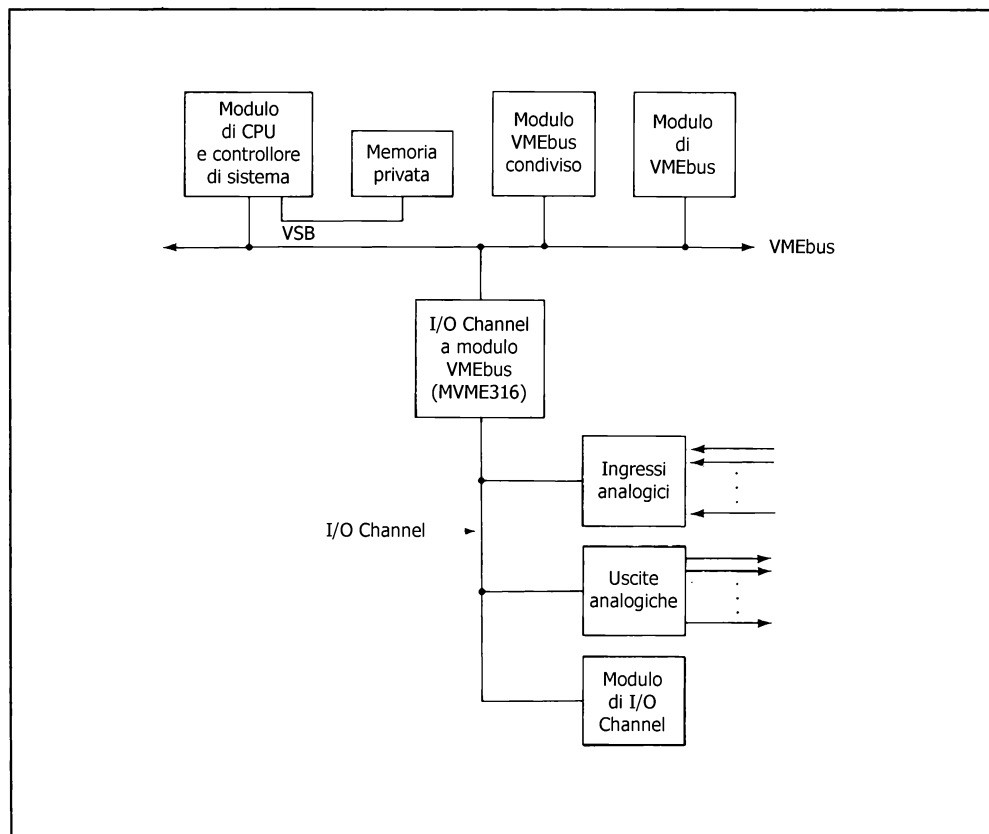



Fig. 14.5 *I/O Channel e VSBbus in un sistema di VMEbus.*

14.2.5 Il VSBbus

La Fig. 14.5 mostra un bus indipendente connesso al modulo della CPU con memoria privata.[10] Questo bus è designato come VSB, abbreviazione di *VME Subsystem Bus* (bus di sottosistema VME). La sua funzione è quella di connettere la CPU ad una memoria privata o a dispositivi periferici che non comunicano tramite il VMEbus. Quindi la CPU non dovrà competere con altri potenziali master di bus per l'uso del VMEbus, quando deve comunicare con la sua memoria privata o con delle periferiche. Nella Fig. 14.5, il modulo di VMEbus che condivide di VMEbus con la CPU potrebbe essere un altro modulo di CPU in un sistema di multielaborazione. Il modulo di memoria del VMEbus potrebbe essere condiviso tra i moduli della CPU in un'applicazione siffatta. Il VSBbus è descritto in dettaglio in un certo numero di riferimenti bibliografici relativi a questo capitolo, elencati nell'App. E alla fine del libro.

[10] Questo bus non dovrebbe essere confuso col "bus locale" incorporato in un modulo di VMEbus. Il VSB serve per la comunicazione tra i moduli.

ESERCIZI

14.2.1 Si definiscano i vantaggi o le finalità delle seguenti caratteristiche del VMEbus:

(a) Trasferimento di dati asincrono rispetto alla temporizzazione della CPU.

(b) Impiego di indirizzi di 24 bit o di 32 bit.

(c) Impiego di una piastra di controllore di sistema separata e di una piastra di CPU anziché di un'unità combinata.

(d) Uso di sei linee di segnale di "modificatore d'indirizzo", anziché di tre linee di segnale di codice di funzione dell'MC68020 per definire il tipo di accesso al bus.

(e) Uso della linea di segnale ACFAIL*.

(f) Uso di 64 piedini del connettore P2 per scopi "definiti dall'utente".

14.2.2 Si supponga che una CPU esegua un'istruzione e richieda l'uso del VMEbus. Si descriva la sequenza di operazioni ed i segnali dell'MC68020 e del VMEbus quando più moduli possono assumere la funzione di master di bus. Si considerino i seguenti casi:

(a) Il bus non è utilizzato.

(b) Il bus è occupato.

(c) Un altro master di bus potenziale richiede il bus mentre la CPU sta eseguendo l'istruzione ed utilizzando il VMEbus.

14.2.3 Si descriva l'utilizzazione del VMEbus quando una CPU sta eseguendo operazioni di I/O come quelle descritte nel par. 13.2. Si considerino i seguenti casi:

(a) I/O programmato.

(b) Trasferimento di DMA.

Perché sono necessari entrambi i tipi di trasferimento sul VMEbus?

14.2.4 L'impiego di un bus di sistema, di un bus privato, e di un bus di I/O separato è stato per lungo tempo una prerogativa dei sistemi di computer di alte prestazioni. Si descrivano i vantaggi e gli svantaggi di un sistema dotato di VMEbus, di VSBbus e di un bus di I/O Channel rispetto ad un sistema con un singolo bus.

14.2.5 S'illustri la possibile corrispondenza tra le linee di segnale dell'MC68020 ed i segnali del VMEbus se la CPU dev'essere connessa al VMEbus.

14.3 I MODULI ED IL SOFTWARE DEL VMEbus

Un modulo di VMEbus è una piastra a circuito stampto che dispone delle specifiche fisiche e della circuiteria d'interfacciamento per la connessione al VMEbus. In generale, un modulo ha uno specifico scopo funzionale, per quanto diverse funzioni possano essere combinate su una singola piastra. Per esempio, il modulo MVME133, presentato nel par. 1.1 e descritto nel par. 13.2, è sia un controllore di sistema di VMEbus che un modulo della CPU MC68020. Tali moduli saranno discussi in questo paragrafo. In primo luogo, saranno presentati vari moduli offerti dalla Motorola e da altri produttori; dopodiché, saranno descritte le varie componenti del software. Le routine prese in considerazione sono fornite dal produttore del particolare modulo per facilitare la programmazione quando tale modulo è impiegato in un'applicazione. Le categori generali di componenti del VMEbus sono definite nella Tab. 14.4.

14.3.1 Esempi di moduli di VMEbus

Sono disponibili moduli di VMEbus per consentire ad un progettista di sistema di costruire un sistema di VMEbus che soddisfi i requisiti di una particolare applicazione. Una volta che è stata presa una decisione in merito ai tipi di moduli necessari ed alle loro caratteristiche, un progettista potrà acquistare i moduli adatti dalla Motorola o da un altro produttore. Questi moduli che sono "disponibili in

Tab. 14.4 *Componenti del sistema VMEbus.*

Componente	Scopo
Moduli di VMEbus, subrack e hardware simile	Costruire un computer con VMEbus per soddisfare esigenze particolari.
Sistema operativo	Controllare l'attività complessiva del computer.[1]
Driver di dispositivo	Controllare una particolare modulo di VMEbus per eseguire i trasferimenti di I/O, la gestione delle interruzioni, e le relative funzioni.
Software applicativo	Eseguire in tutto o in parte le operazioni richieste da un'applicazione.

Nota 1: Il sistema operativo comprende solitamente dei programmi per lo sviluppo del software, quali editor, assemblatori e compilatori, come pure dei programmi di ausilio al debugging.

commercio" soddisfano i requisiti generali di un sistema per le funzioni ordinarie. Se è necessario un modulo per soddisfare un requisito particolare, esso potrà essere progettato, costruito e collaudato attenendosi alle specifiche del progettista del sistema.

Nella Tab. 14.5 viene fornito un elenco parziale dei moduli prodotti dalla Motorola. Per ognuno di essi, sono elencati il tipo e la designazione della Motorola, per dare al lettore un'idea della varietà di moduli che si possono acquistare. Sono compresi i moduli di controllore di sistema, i moduli di CPU e di memoria, i controllori per dispositivi periferici, nonché un certo numero di moduli per altri scopi.

Tab. 14.5 *Moduli di VMEbus.*

Tipo di modulo	**Codice della Motorola**[1]
Controllore di sistema	MVME050
Moduli di CPU	MVME132, MVME133
Memoria	MVME202, MVME225
Controllori di dispositivi periferici	
Disco Winchester	MVME319
o disco flessibile	MVME320, MVME321
Vari	
Collegamento in rete	MVME330
Bus IEEE488	MVME300[2]
I/O Channel	MVME316[3]
MAP	MVME372
Display grafico	MVME390
Trasferimento seriale e parallelo	MVME335, MVME340

Note:

1. I moduli elencati rappresentano soltanto alcuni dei molti disponibili dalla Motorola.

2. Il bus IEEE488 è usato per comunicazioni tra gli strumenti di laboratorio.

3. MAP (*Manufacturing Automation Protocol:* protocollo di automazione della produzione) è una specifica di un gruppo di standard per consentire la comunicazione tra apparecchiature di produttori diversi. Il suo impiego tipico è nella specifica dell'interfaccia e del protocollo di comunicazione in applicazioni industriali e di robotica. La piastra dispone di un'interfaccia tra il VMEbus e la rete locale di MAP.

Il modulo del controllore di sistema (MVME050) agisce da controllore di VMEbus in un sistema con VMEbus in cui altri moduli possono svolgere la funzione di master di bus. Per esempio, un modulo controllore di sistema potrebbe essere utilizzato un un sistema multiprocessore contenente vari moduli di CPU. Il modulo MVME 132 di CPU ha un'interfaccia di VMEbus e di VSBbus, ma non può agire come controllore di sistema. Quindi un modulo MVME050 o equivalente sarebbe necessario in un sistema contenente un MVME132. Il modulo MVME133 di CPU contiene il controllore di sistema sulla propria piastra.

I moduli di CPU elencati nella Tab. 14.5 contengono un bus locale ed una memoria locale, come pure un'interfaccia di VMEbus. Un bus locale consente al modulo di CPU di eseguire programmi senza l'impiego del VMEbus. Una CPU può anche effettuare accessi al VMEbus indirizzando altri moduli nel sistema, ad esempio un modulo di memoria. Ciascun modulo può contenere un programma di monitor nella memoria a sola lettura (ROM) per facilitare il caricamento del programma dall'unità di memoria a disco ed altre operazioni. Oltre ai moduli basati sull'MC68020 qui descritti, la Motorola ed altri produttori offrono moduli di CPU che impiegano il processore MC68030.

I moduli di accesso alla memoria elencati nella Tab. 14.5 sono provvisti di una memoria di lettura/scrittura di tipo RAM (*Random Access Memory:* memoria ad accesso casuale) per contenere programmi e dati per un modulo di CPU che impiega il VMEbus. I moduli di memoria differiscono per il tipo di RAM che contengono e per il numero di byte di memoria disponibili. Un progettista seleziona un modulo in base alla durata del ciclo di memoria (lettura/scrittura) ed alla dimensione in byte necessaria. Un modulo tipico ha una capacità di memorizzazione compresa tra 512 KB e 2 MB.

Nella Tab. 14.5 è elencato un certo numero di moduli per il controllo di periferiche. Tali moduli controllano sia le unità a disco flessibile (*floppy disk*) che le unità a disco rigido (*hard disk*) di tipo Winchester. Sotto il controllo di un sistema operativo, questi moduli di controllo di periferiche servono ad effettuare i trasferimenti di DMA tra un modulo di memoria e l'unità a disco.

Altri moduli elencati nella Tab. 14.5 svolgono varie funzioni in un sistema con VMEbus. Alcuni di essi consentono di connettere il sistema VMEbus ad una rete (MVME330) o ad un altro sistema di bus (MVME300, MVME372). Un modulo di display grafico è utilizzato per controllare uno schermo di visualizzazione per applicazioni avanzate di grafica su computer. Sono disponibili anche moduli che eseguono trasferimenti di I/O seriali e paralleli verso unità periferiche. Tutti questi moduli richiedono programmi appropriati per controllare le loro operazioni in un'applicazione specifica.

I moduli di VMEbus di altri produttori. La Tab. 14.6 elenca alcuni moduli di VMEbus disponibili da produttori diversi dalla Motorola. Si tratta di un elenco molto ridotto rispetto alle centinaia di moduli VMEbus disponibili in commercio. In

Tab. 14.6 *Moduli di VMEbus.*

Tipo di modulo	Produttori
Moduli di CPU	
Modulo di MC68030	Plessey Microsystems, Pearl River, N.Y.
Moduli della famiglia M68000[1]	Force Computers Inc., Los Gatos, California
Funzioni varie	
I/O analogico e digitale	Burr-Brown Corporation, Tucson, Arizona
Elaborazione di segnali digitali	Ironics Inc., Ithaca, N.Y.
Controllore di grafica	IO, Inc., Tucson, Arizona
Modulo per la realizzazione di prototipi	XYCOM, Inc., Saline, Michigan

Nota 1: La famiglia dell'M68000 denota qui entrambe le famiglie di processori MC68000 e MC68020.

effetti, la maggior parte delle aziende elencate nella tabella producono o commercializzano sistemi completi con VMEbus, inclusi vari tipi di subrack di VMEbus e di sistemi operativi per computer con VMEbus. Altri produttori non elencati producono moduli di VMEbus per applicazioni particolari, quali l'elaborazione di immagini o l'elaborazione di array. Si rimanda il lettore ai riferimenti bibliografici relativi a questo capitolo, risportati nell'App. E alla fine del libro, per ulteriori informazioni concernenti i prodotti relativi al VMEbus.

La maggior parte delle società che offrono prodotti di VMEbus vendono moduli di CPU e moduli di memoria. I moduli differiscono nel costo e nelle prestazioni. Per esempio, la Plessey produce un modulo di CPU che impiega un processore MC68030. La Force Computers offre una gamma completa di moduli di CPU per la famiglia di processori a 16 bit e a 32 bit. Entrambe queste società offrono anche altri moduli che potrebbero essere inclusi con quelli della Tab. 14.6.

14.3.2 Considerazioni di software per i moduli di VMEbus

La maggior parte dei moduli di VMEbus discussi finora operano sotto il controllo del programma per svolgere qualche funzione utile in un sistema di computer. Il modulo della CPU, per esempio, esegue un programma applicativo che ha lo scopo di mettere il sistema del computer in grado di svolgere una funzione specifica. Durante la sua esecuzione, il programma applicativo può chiamare a sua volta il sistema operativo per completare un compito specifico, come il trasferimento dei dati elaborati al terminale video dell'operatore. Il programmatore di applicazioni di solito chiama una routine del sistema operativo per effettuare il trasferimento, al fine di evitare il compito tedioso di programmazione del chip di I/O o del modulo, come spiegato nel par. 7.6. In effetti, nei computer con VMEbus dotati di un sistema operativo, tutti i trasferimenti di I/O, la gestione delle interruzioni, ed operazioni simili sono eseguiti da routine del sistema operativo. La Fig. 14.6 mostra una sequenza tipica di eventi per un trasferimento di I/O.

Il programma applicativo causa un trasferimento di I/O richiedendo un servizio del sistema operativo. Nei sistemi basati sull'MC68020, ciò avviene solitamente

Programma applicativo

Sistema operativo

Esecuzione normale

Richiesta per trasfer. di I/O

TRAP

Driver di dispositivo
- Prepara il trasferimento di dati (I/O programmato, pilotato da interruzione, o in DMA)
- Gestisce le interruzioni o altre eccezioni

RTE

Continua l'esecuzione normale

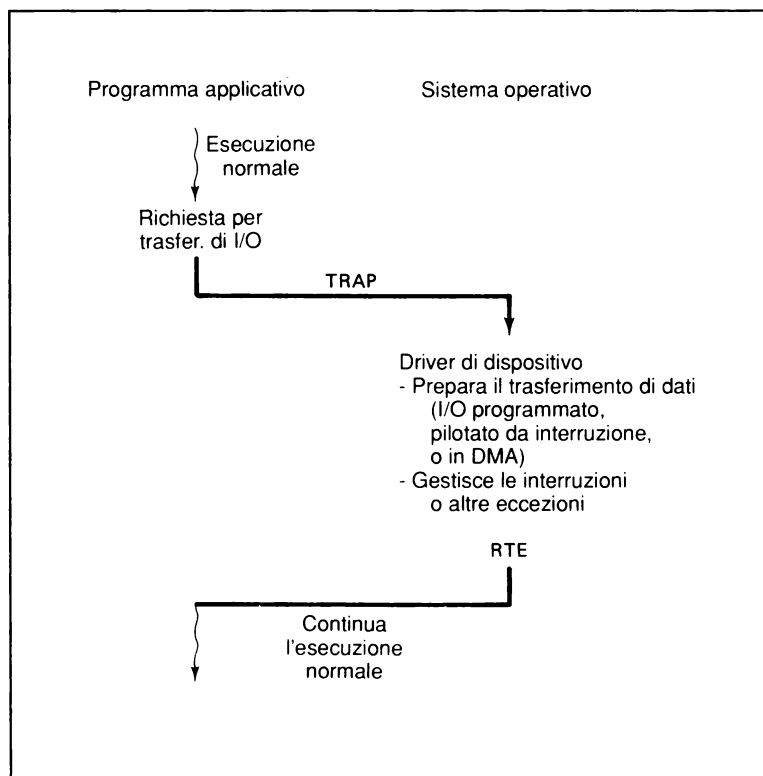



Fig. 14.6
Tipiche operazioni di software durante un trasferimento di I/O.

eseguendo un'istruzione TRAP. Il programma richiedente deve specificare gli indirizzi di sorgente e di destinazione dei dispositivi interessati ed il numero di byte da trasferire. Tale richiesta avvia l'esecuzione di una serie di routine che inizializzano il trasferimento dei dati sul VMEbus e gestiscono qualsiasi interruzione o eccezione relativa al trasferimento di I/O. Nella terminologia dei sistemi operativi della Motorola, tali routine costituiscono nel loro insieme il cosiddetto *driver del dispositivo.* La discussione presente suddivide i programmi per il controllo del VMEbus in quelli che fungono da sistema operativo, inclusi i relativi driver di dispositivi, ed altri che rappresentano programmi applicativi. Molte società forniscono entrambi i tipi di software. Tali società comprendono la maggior parte dei produttori di moduli di VMEbus.

Il software di sistema per i moduli VMEbus. A parte i sistemi più semplici, i computer con VMEbus sono controllati da un sistema operativo che provvede al caricamento nella memoria del programma contenuto nell'unità a disco, ai trasferimenti di I/O, alle routine di gestione delle eccezioni, e ad operazioni simili. I sistemi operativi più diffusi per computer con VMEbus basati sull'MC68020 comprendono il VERSAdos della Motorola ed il V/68 presentati nel par. 1.2. Questi sistemi operativi ed alcuni altri sono elencati nella Tab. 14.7. Tuttavia, il numero di società che attualmente creano un sistema operativo per computer VMEbus è relativamente piccolo rispetto alle dozzine di produttori di moduli VMEbus. Quelle società che si concentrano soprattutto sulla produzione di hardware possono offrire un sistema operativo prodotto da un'altra azienda. Per esempio, La Force Computers offre il sistema operativo Eyring PDOS coi suoi prodotti. In ogni caso, quando un nuovo modulo viene aggiunto al sistema di VMEbus, il sistema operativo deve essere in grado di gestire il suo driver di dispositivo, se il funzionamento del modulo richiede i servizi del sistema operativo.

Il *driver di dispositivo* per un modulo VMEbus consiste di tutte le routine necessarie per consentire al modulo di funzionare come parte del sistema VMEbus. Un insieme fondamentale di routine dovrebbe includere routine selezionate per la gestione delle eccezioni, come descritto nel cap. 11. Una di tali routine potrebbe

Tab. 14.7 *Sistemi operativi per computer con VMEbus basati sull'MC68020.*

Sistema operativo	Produttore
VERSAdos, V/68	Motorola, Inc.
PDOS	Eyring Research Institute, Inc., Provo, Utah
pSOS	Software Components Groups, Inc., Santa Clara, California
Sistemi operativi di tipo UNIX	Varie società

gestire l'eccezione di TRAP per accettare le richieste di I/O. Altre routine potrebbero gestire interruzioni ed errori possibili indicati da un segnale di errore di bus proveniente dal modulo.

Quando i produttori offrono dei moduli di VMEbus, il produttore del modulo fornisce talvolta un driver di dispositivo per un modulo come parte del sistema operativo offerto da quel produttore.[11] In altri casi, un produttore del modulo può fornire vari driver di dispositivo per un modulo di VMEbus. I diversi programmi di driver di dispositivo sono concepiti per essere compatibili coi vari sistemi operativi. Per esempio, la Burr-Brown fornisce i driver di dispositivo di pSOS e VERSAdos per molti dei suoi moduli di I/O. Quindi, la selezione di un sistema di VMEbus, insieme con i driver di dispositivo appropriati per la compatibilità col sistema operativo che si intende utilizzare, richiede un'attenta valutazione se i moduli di diversi produttori devono essere integrati in un sistema.

Programmi applicativi per moduli di VMEbus. Alcuni moduli di VMEbus possono richiedere grandi e complessi programmi per consentire al sistema di soddisfare i requisiti di un'applicazione. Questi programmi applicativi sono solitamente creati e collaudati come parte dell'impegno di sviluppo del software di sistema, come definito nel par. 1.2. Tuttavia, in certi casi, almeno una porzione del software applicativo svolge le operazioni ordinarie. Ciò è particolarmente vero quando l'applicazione richiede operazioni matematiche quali la moltiplicazione matriciale o l'analisi di Fourier. Per esempio, la Ironics, Inc. produce un modulo di elaborazione di segnali digitali che compariva nell'elenco della Tab. 14.6. La Ironics fornisce anche una certa varietà di routine applicative affinché il modulo possa eseguire le trasformate di Fourier ed operazioni simili. Il lettore interessato potrà consultare i cataloghi dei produttori per determinare il software applicativo disponibile per un particolare modulo.

ESERCIZI

14.3.1 Si scelga un produttore di componenti VMEbus e si descrivano i prodotti offerti da tale società. Si includano considerazioni sia sull'hardware che sul software.

14.3.2 Si descrivano gli aspetti che devono essere considerati quando si seleziona un sistema di VMEbus. Si definisca a grandi linee il processo di selezione e si risponda alle seguenti domande:

(a) Qual è l'ordine migliore per selezionare componenti quali il modulo di CPU ed il sistema operativo?

(b) Quali sono i vantaggi e gli svantaggi di acquistare componenti da diversi produttori anziché acquistare un sistema completo da una sola società?

---

[11] Se un driver di dispositivo non è disponibile dal produttore del modulo, il programmatore del sistema deve progettare e provare le routine del driver. Tali routine sono quindi incorporate nel sistema operativo per il computer.

14.3.3 Si descriva il driver di dispositivo per un modulo controllore di DMA in termini delle routine di gestione delle eccezioni necessarie in un sistema di VMEbus basato sull'MC68020.

14.3.4 Si determini il modo in cui un nuovo driver di dispositivo viene aggiunto al sistema operativo che si sta utilizzando. (Ciò potrebbe richiedere una lettura approfondita del manuale per l'utente e di altra documentazione del sistema operativo.)

14.4 SISTEMI DI VMEbus

La Fig. 14.7 mostra il diagramma a blocchi di un sistema VMEbus di esempio, che incorpora alcuni moduli ed il sistema operativo VERSAdos della Motorola. I moduli sono l'MVME133 di CPU, il modulo di memoria MVME225-2, ed il modulo MVME320 controllore di unità a disco. Questi moduli e l'unità a disco stessa (MVME822) possono essere alloggiati in un subrack di VMEbus a nove slot, come illustrato nella Fig. 14.8. L'unità a disco è inserita nel subrack a destra degli slot di schede, ma non è connessa direttamente al VMEbus. Un sistema siffatto potrebbe essere già completo, a parte il cavo di alimentazione e la connessione al terminale dell'operatore. C'è un collegamento di comunicazione ad un computer *host* nel modo descritto nel par. 5.1. Tale collegamento è usato quando si utilizza il cross-assemblatore del computer host per lo sviluppo del programma e la piastra MVME133 è sotto il controllo del monitor 133BUG. Quando il sistema operativo VERSAdos controlla il sistema del computer, il monitor viene utilizzato solamente all'inizio per caricare i programmi dall'unità a disco. Il VERSAdos contiene un assemblatore residente e dell'altro software per lo sviluppo dei programmi.[12] In questo paragrafo, saranno descritti i moduli di VMEbus, la mappa d'indirizzi ed il software di sistema per il computer con VMEbus dell'esempio.

14.4.1 I moduli di VMEbus in un sistema di esempio

Il computer con VMEbus in esame, illustrato nella Fig. 14.7, consiste di vari moduli, di un'unità a disco, e del sistema operativo VERSAdos. La Tab. 14.8 elenca le caratteristiche dei moduli e dell'unità a disco. Qui sono descritte tali caratteristiche e la loro importanza per il sistema.

Il modulo della CPU ed il controllore di sistema. L'MVME133 è stato descritto nel par. 13.2, come esempio di un computer su piastra singola, per effettuare i trasferimenti seriali ad unità esterne ed altre operazioni. In particolare, il chip periferico multifunzione MC68901 è stato descritto ed impiegato negli esempi per i trasferimenti di I/O. Quelle operazioni di trasferimento di dati non richiedevano l'uso del VMEbus, poiché tali trasferimenti avvenivano sul bus di CPU locale. Nella presente discussione, sarà posta in evidenza l'interazione del modulo della CPU con VMEbus.

[12] Il software di sistema per lo sviluppo di programmi è stato presentato nel par. 1.2. Le differenze tra un crosso assemblatore ed un assemblatore residente sono state discusse nel par. 5.1. il cross-assemblatore è utilizzato da studenti che possono disporre di un computer host VAX o di un PC IBM.

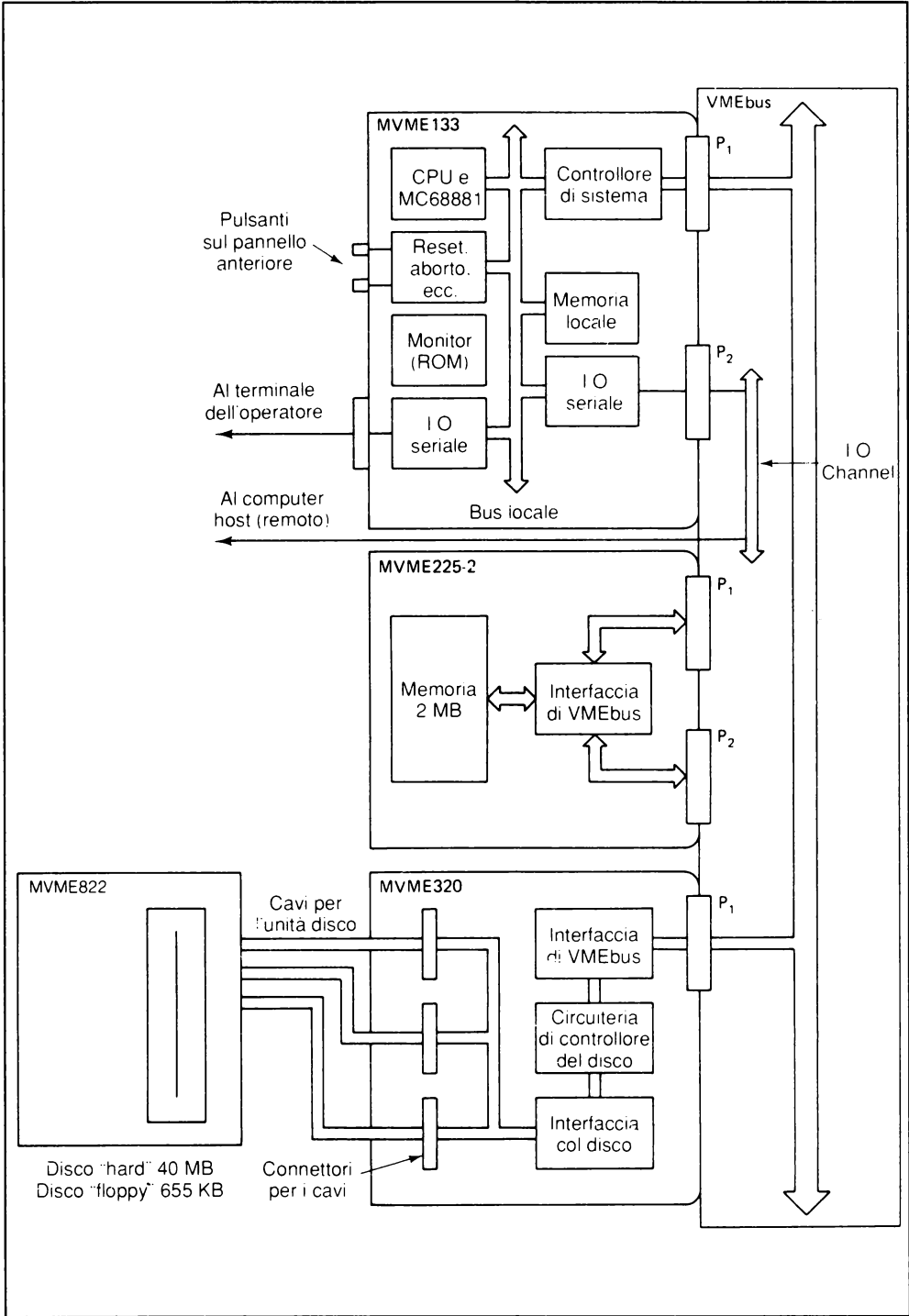
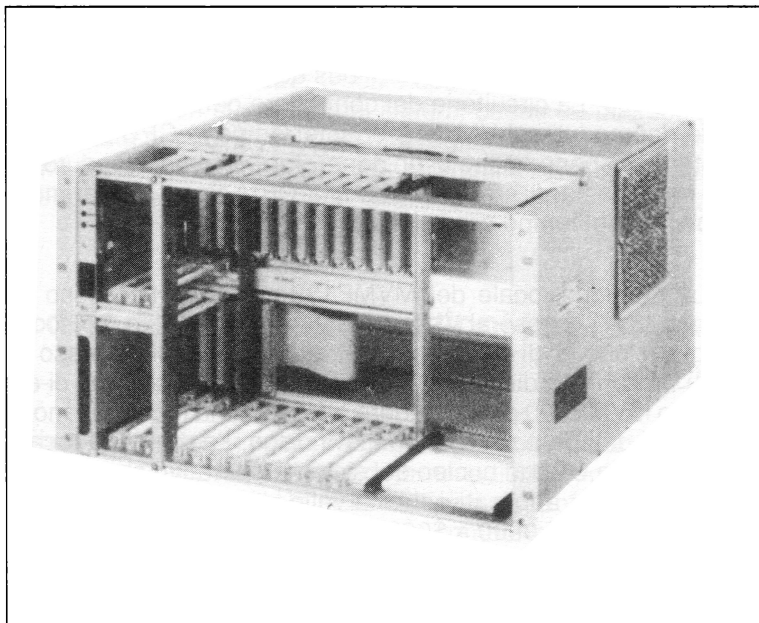



Fig. 14.7 *Diagramma a blocchi di un sistema VMEbus di esempio.*

Fig. 14.8
Subrack di VMEbus (MVME943). (Per gentile concessione di Motorola, Inc.)

Tab. 14.8 *Moduli di VMEbus.*

Modulo	Caratteristiche
CPU: MVME133	CPU MC68020 e coprocessore MC68881; Clock a 12.5 MHz; 1 MB di memoria locale, accessibile dal VMEbus; Porte seriali e tre timer (MC68901); Monitor 133BUG (opzionale). Controllore di sistema per sistemi VMEbus.
Memoria: MVME225-2	2 MB di memoria (RAM); Tempo di lettura di 300 ns.
Controllore del disco: MVME320	Può controllare fino a 4 unità; Due unità a disco flessibile (*floppy*); Due unità a disco rigido (*hard*) di tipo Winchester.
Unità a disco: MVME822	Unità a disco Winchester da 40 MB, $5^1/_4$ pollici; Unità a disco flessibile da 655 KB, $5^1/_4$ pollici.

Come controllore di sistema, il modulo MVME133 controlla l'accesso al VMEbus.[13] Conformemente alle priorità dei vari moduli, la circuiteria di controllore dell'MVME133 concede l'accesso al bus quando la propria CPU o un altro modulo ne richiede l'uso. La circuiteria del controllore del sistema gestisce le interruzioni generate dai moduli sul VMEbus ed interrompe la CPU all'occorrenza. Il controllore di sistema dispone anche di un generatore di tempo scaduto (*watchdog timer*), che invia un segnale di errore di bus alla CPU se un trasferimento di dati non viene completato nel tempo prestabilito.

La memoria locale dell'MVME133 consente l'accesso alla CPU mediante il bus locale o l'accesso al VMEbus da un altro modulo. Lo scopo di questo accesso multiplo è quello di consentire il trasferimento di accesso diretto alla memoria (*DMA*) tra l'unità a disco e la memoria locale. Nel sistema di esempio, il controllore del disco MVME320 può accedere alla memoria locale sul modulo di MVME133 per trasferire i dati da e verso un'unità a disco. La memoria locale contiene la tabella di vettori della CPU, il nucleo del sistema operativo che è residente nella memoria, ed i programmi applicativi. L'intervallo d'indirizzamento della memoria locale si estende da $0000 0000 a $000F FFFF.

Il modulo di memoria. L'MVME225-2 è un modulo di memoria da 2 MB che contiene i byte delle locazioni di memoria $100000-$2FFFFF nel sistema. Questo modulo di memoria di VMEbus potrebbe contenere programmi o dati che non sono contenuti nella memoria locale del modulo MVME133.

Il controllore dell'unità a disco e i dischi. Il controllore dell'unità a disco si comporta come controllore di DMA per trasferire le informazioni tra le unità a disco e la memoria locale o di VMEbus. Esso opera sotto il controllo del sistema operativo VERSAdos per effettuare i trasferimenti. L'unità a disco MVME822 contiene sia un'unità a disco flessibile che un'unità a disco rigido (Winchester).

Nell'MVME822, un disco flessibile da $5^1/_4$ pollici, denominato *floppy disk*, serve a contenere le informazioni che non sono memorizzate permanentemente nel sistema.[14] Questo tipo di disco svolge varie funzioni importanti nel sistema. Per esempio, il sistema operativo viene caricato per la prima volta da dischi flessibili sul disco Winchester. Ciò avviene durante il processo di generazione del sistema, che sarà descritto nel par. 14.5. Inoltre, poiché un disco flessibile può essere rimosso dalla sua unità, esso può essere usato per trasferire informazioni tra due sistemi indipendenti che dispongono di unità a disco flessibile.[15]

[13] L'MVME133 viene selezionato come controllore di sistema mediante un "ponticello" sulla piastra del modulo. Un ponticello (*jumper*) è un pezzetto di filo conduttore che viene usato per selezionare un particolare attributo del modulo.

[14] Un disco di queste dimensioni è talvolta denotato come *minifloppy*, poiché il diametro dei dischi flessibili originali era di $8^1/_2$ pollici. Esistono anche dischi flessibili di diametro inferiore (cioè, di $3^1/_2$ pollici).

[15] Le dimensioni meccaniche ed il formato devono essere i medesimi per le unità a disco dei due sistemi. Il formato descrive la configurazione delle informazioni sul disco.

Un altro tipo di disco, denominato disco rigido (*hard disk*) o Winchester, fa parte dell'unità a disco del modulo MVME822. Questo tipo di disco è stato sviluppato dall'IBM ed è stato impiegato con successo nei sistemi basati su microcomputer. L'unità Winchester è sigillata per cui il suo mezzo di registrazione non è rimovibile. In genere, questa unità serve per memorizzare i programmi e i dati che costituiscono la parte più "stabile" del sistema, rispetto alle informazioni memorizzate sui dischi flessibili.

14.4.2 Mappa di indirizzi ed altre opzioni per i sistemi di VMEbus

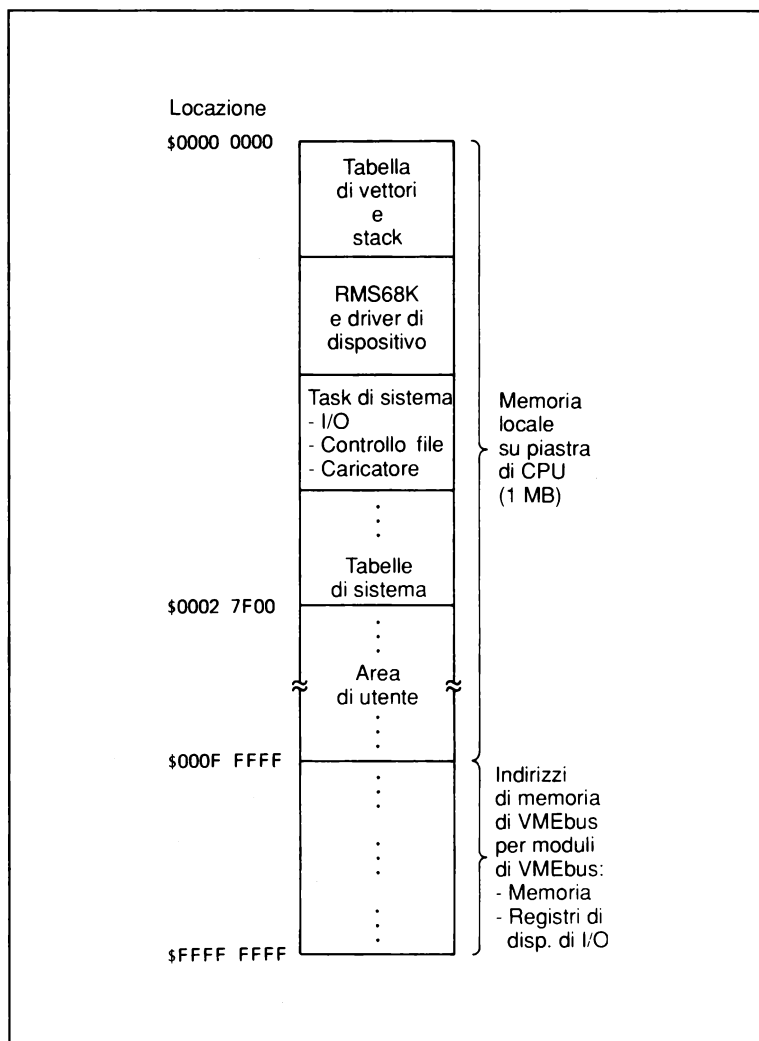
I moduli di VMEbus ed il sistema operativo VERSAdos offrono ad un progettista di sistema un alto grado di flessibilità nella determinazione degli indirizzi per i moduli e delle loro priorità sul VMEbus. Un sistema tipico potrebbe avere la configurazione di memoria illustrata nella Fig. 14.9. Ogni componente del sistema, incluso il software, può essere definito da una mappa d'indirizzi siffatta, che illustra la sua ubicazione nel sistema. Poiché i sistemi basati sull'MC68020 impiegano l'I/O rappresentato nella memoria, ogni locazione di memoria ed ogni registro su un modulo di VMEbus avranno un indirizzo unico nel sistema. Queste locazioni sono accessibili da parte delle istruzioni dell'MC68020, come l'istruzione MOVE.

Un sistema tipico ha la porzione residente del sistema operativo, denominata "nucleo" (*kernel*), agli indirizzi di memoria inferiori, come mostrato nella Fig. 14.9. La tabella di vettori dell'MC68020 e lo stack di sistema sono agli indirizzi più bassi. RMS68K è la porzione residente del sistema operativo VERSAdos.[16] Esso comprende i driver di dispositivo, i programmi di sistema (task), e le tabelle di sistema. I compiti di sistema svolgono le funzioni di I/O, controllano la struttura dei file nelle tabelle di sistema e sul disco, e caricano o memorizzano i programmi utilizzando l'unità a disco del computer. Un'area di memoria di utente è mostrata sopra lo spazio del sistema operativo nella memoria locale. La memoria del VMEbus è situata oltre l'intervallo di indirizzi della memoria locale sulla piastra della CPU. Quindi, le istruzioni della CPU distinguono tra la memoria locale ed il VMEbus, indirizzo per indirizzo. Ad esempio, la sequenza di istruzioni

```
MOVE.W    D0,$0002FFF0      ; memoria locale
MOVE.W    D1,$002FFFF0      ; VMEbus
```

trasferisce una word alla memoria locale ed una word alla memoria del VMEbus, rispettivamente, usando l'intervallo di indirizzi mostrato nella FIg. 14.9. Altre caratteristiche di un modulo, come la sua priorità d'interruzione, sono selezionate da ponticelli o interruttori sul modulo. La circuiteria del controllore di sistema interpreta queste priorità ed accoglie la richiesta d'interruzione del modulo di massima priorità, qualora più richieste si presentino simultaneamente. Si rimanda il lettore

[16] RMS68K è l'acronimo di *Real-time Multitasking Software for the MC68000.*

Fig. 14.9
Un esempio di mappa di indirizzi per un sistema di VMEbus.

al manuale dell'utente di un particolare modulo per determinare i passi richiesti nell'assegnazione delle interruzioni e delle priorità delle richieste di bus. Una discussione delle considerazioni di progetto del sistema per la selezione delle priorità sarà presentata nel par. 14.5.

14.4.3 Il software di sistema per i sistemi di VMEbus

Il sistema operativo in un computer di VMEbus controlla tutte le operazioni del sistema che riguardano direttamente l'hardware. Il VERSAdos (*VERSAtile Disk*

Operating System) della Motorola controlla tutti questi aspetti delle operazioni del computer e fornisce anche il software per lo sviluppo del programma.[17] La sua struttura fondamentale è mostrata nella Fig. 14.10. Nella figura, lo strato più interno consiste di RMS68K. Questo nucleo controlla la gestione delle eccezioni e le richieste di I/O avanzate dai programmi applicativi. Questi programmi sono denominati "*user task*" (task o compiti di utente) nella terminologia della Motorola. L'area di "espansione di utente" (*user expansion*) nella figura indica che i driver del dispositivo ed altre routine possono essere aggiunte a questo livello. Per la maggioranza dei moduli di VMEbus della Motorola, i driver di dispositivo fanno parte del modulo RMS68K.

Il livello successivo al di sopra dell'RMS68K controlla la struttura di file per il sistema operativo, incluse le informazioni nell'unità a disco. I programmi sono eseguiti in accordo con gli algoritmi impiegati nella sezione di amministrazione multiutente del livello successivo. I programmi *batch* sono i programmi eseguiti ad un livello di priorità inferiore a quello degli altri task. Uno *spooler* provvede al controllo ordinato delle operazioni di I/O per un'unità di I/O condivisa da più utenti.[18]

Il software di sistema per lo sviluppo di programmi è considerato come situato al livello più esterno nella struttura del sistema operativo. Questi programmi risiedono nell'unità di memoria a disco e vengono caricati su richiesta del programmatore del computer.

Un diagramma più dettagliato nella Fig. 14.11 illustra il modo in cui un task di utente chiama una delle routine fondamentali di RMS68K o del manager di file e del programma caricatore (*loader*). Per richiedere il servizio desiderato, viene eseguita l'appropriata istruzione TRAP #N. Le informazioni passate alla routine del sistema operativo dipendono dallo scopo. I protocolli per tali richieste sono definiti nell'*User's Manual* del VERSAdos.[19] In generale, le istruzioni del programma devono definire innanzitutto i parametri appropriati per le operazioni, come richiesto dalla routine di sistema prima che venga eseguita l'istruzione TRAP. Quando il servizio viene eseguito, il sistema operativo riporta il controllo al task di utente richiedente, se necessario.

Dunque la programmazione di un modulo VMEbus per eseguire i trasferimenti di I/O o altre operazioni richiede semplicemente le chiamate appropriate al sistema operativo se il driver del dispositivo per il modulo fa parte del VERSAdos. Il programmatore deve soltanto conoscere le caratteristiche del modulo al fine di

[17] Si possono aggiungere routine speciali al sistema operativo per controllare i moduli che non fanno parte del sistema standard.

[18] SPOOL è l'acronimo di *Simultaneous Peripheral Output On Line:* uscita simultanea di periferiche sulla linea. Il software consente a programmi eseguiti in modo concorrente di condividere un'unità periferica.

[19] Alcuni esempi di chiamate di sistema sono stati forniti nel par. 7.6 per routine che chiamano il monitor 133BUG. I princìpi sono simili a quelli di una chiamata al sistema operativo.

Nota: RMS68K è il *kernel* (nucleo) di software multitasking in tempo reale del sistema operativo VERSAdos.

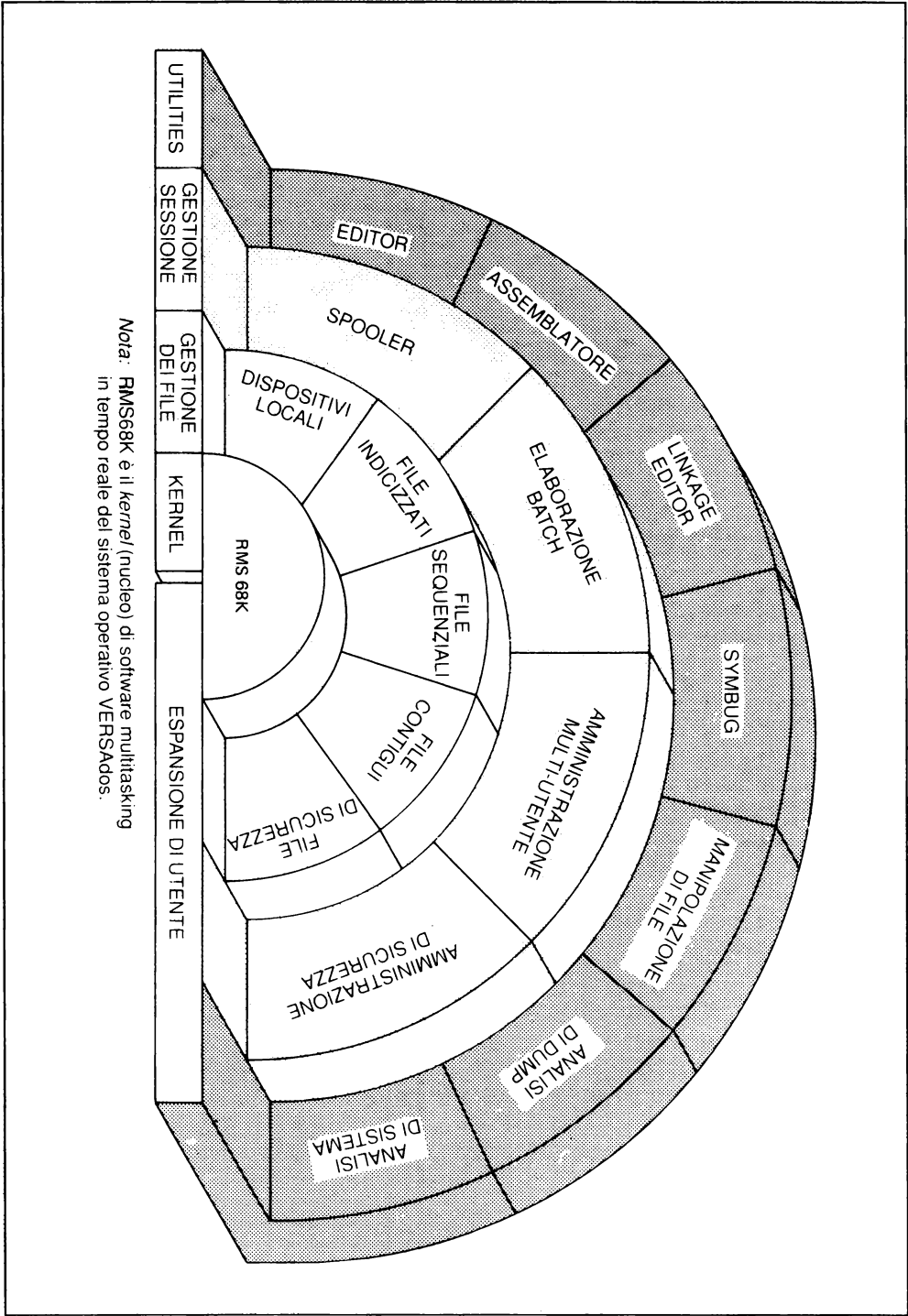



Fig. 14.10 *La struttura del VERSAdos. (Per gentile concessione di Motorola, Inc.)*

STRATI DEL VERSAdos

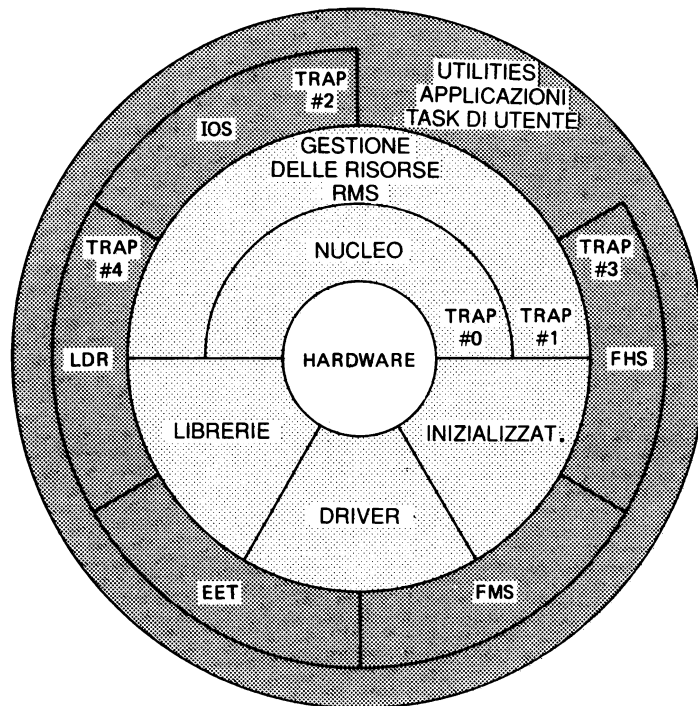
UTILITIES
APPLICAZIONI
TASK DI UTENTE
TRAP #2
IOS
GESTIONE DELLE RISORSE RMS
TRAP #4
NUCLEO
TRAP #3
TRAP #0
TRAP #1
LDR
HARDWARE
FHS
LIBRERIE
INIZIALIZZAT.
DRIVER
EET
FMS

Note:
(1) IOS è la routine di gestione dell'I/O.
(2) LDR è il caricatore (*loader*).
(3) FMS e FHS sono routine per la gestione dei file.

Fig. 14.11 *Richieste di utente a RMS68K. (Per gentile concessione di Motorola, Inc.)*

controllarne il funzionamento.[20] Inoltre, il sistema operativo coordina le richieste per l'uso di un modulo quando più task lo richiedono contemporaneamente. Tali richieste sono comuni in un sistema multiutente in cui una stampante parallela è condivisa da più utenti.

[20] Per esempio, il programmatore non ha bisogno di inizializzare un dispositivo per il trasferimento di I/O quando esso dispone di un driver di dispositivo. Le istruzioni del programma devono definire soltanto le locazioni dei dati da trasferire, il numero di byte e la direzione del trasferimento.

ESERCIZI

14.4.1 Si descriva la procedura per aggiungere i seguenti nuovi moduli ad un sistema di VMEbus esistente:

(a) Un modulo di memoria (2 MB)

(b) Un modulo di stampante seriale (il VERSAdos ha un driver di dispositivo per questo modulo)

(c) Un altro modulo di CPU per creare un sistema multiprocessore.

14.4.2 Qual è il numero approssimativo di unità a disco flessibile necessarie per memorizzare in un'unità MVME822 la quantità di informazioni contenibile in un disco Winchester?

14.5 PROGETTAZIONE ED AVVIAMENTO DEL SISTEMA

Questo paragrafo inizia presentando una breve descrizione della progettazione del sistema e del processo di selezione. Lo scopo è quello di rivedere alcuni criteri che devono essere seguiti per consentire al progettista di sistema di definire i requisiti di un sistema VMEbus. Dopodiché, saranno considerati i passi necessari per mettere assieme i vari componenti del sistema VMEbus e per avviarne le operazioni. Queste includono la definizione dei parametri dell'hardware per il sistema e lo svolgimento del processo di generazione del sistema.

14.5.1 Progettazione del sistema e selezione dei moduli

Il processo di progettazione del sistema, introdotto nel par. 2.3, richiede che il progettista specifichi tutti i criteri importanti che consentono al sistema di soddisfare i requisiti di un'applicazione. Una volta che il progetto è stato completato, si dovrebbe scegliere un computer con un sistema operativo che consenta l'acquisto o la creazione di software addizionale, cioè che possa essere aggiunto al computer di base. Naturalmente, ai fini della presente discussione, qui è stato scelto un sistema con VMEbus basato sull'MC68020. La progettazione del sistema è quindi incentrata sulla selezione degli appropriati moduli di VMEbus per soddisfare i requisiti imposti dal computer.

La Tab. 14.9 elenca gli aspetti generali che si dovrebbero prendere in considerazione nella fase di analisi dei requisiti del sistema. Il tipo di sistema determina in una certa misura i criteri che saranno importanti nell'analisi del sistema. Per

esempio, un sistema "dedicato" è un sistema adibito ad un'applicazione specifica: sarebbe impossibile definire i requisiti del sistema se non si comprendesse a fondo l'applicazione. Un sistema multiutente, tuttavia, può essere definito in generale dal numero di utenti, dai tipi di problemi che saranno chiamati a risolvere (se noti), e dal tempo di risposta richiesto a ciascun utente. I sistemi in tempo reale o pilotati da interruzione, utilizzati nelle applicazioni di controllo, hanno in genere un numero relativamente grande di canali di I/O e devono rispondere rapidamente alle interruzioni. Un computer per la programmazione tecnica o scientifica richiede la capacità della virgola mobile ed una notevole quantità di software di sistema, ad esempio vari compilatori di linguaggi ad alto livello. Pertanto, per una certa applicazione, perfino i requisiti più generali possono determinare il tipo ed il numero di moduli di cui un sistema avrà bisogno. Le caratteristiche particolari dei moduli dovranno essere specificate da un'analisi più dettagliata.

Tab. 14.9 *Analisi e selezione del sistema.*

SPECIFICAZIONE	CRITERI	SELEZIONE
Tipo di sistema		
Sistema dedicato	Numero di utenti	Tipo e numero di moduli
Multiutente	Prestazioni	
Tempo reale (pilotato da interruzione)	Elaborazione richiesta	
Programmazione scientifica	Protezione del sistema	
Temporizzazione del sistema		
Velocità di trasferimento su bus	Velocità di elaborazione del sistema	Velocità operativa del VMEbus, del modulo della CPU e dei moduli di memoria
Numero di interruzioni al secondo	Tempo di risposta alle interruzioni	
Velocità di elaborazione	Istruzioni al secondo	
Requisiti di I/O		
Numero di canali	Requisito di memorizzazione in buffer	Tipo, velocità operativa e numero di moduli di I/O
Velocità di trasferimento dei dati	Velocità di memorizzazione dei dati o velocità di uscita in byte al secondo	
Requisiti di memorizzazione		
Memoria (OS, programmi, buffer di I/O)	Capacità di memoria	Numero e dimensioni dei moduli di memoria e tipo di unità a disco
Memorizzazione su disco	Capacità del disco	
Unità periferiche		
Unità di visualizzazione, ingresso di dati o di stampa	Caratteristiche dell'unità	Tipo di modulo del controllore

Si può stimare la temporizzazione del sistema complessivo per una certa applicazione, se i tempi sono definiti esattamente in termini della velocità di trasferimento dei dati, del numero di interruzioni al secondo, e di fattori simili. A seconda dell'applicazione, le stime di temporizzazione possono risultare piuttosto semplici da effettuare o praticamente impossibili da determinare. In ogni caso, alcune stime devono essere fatte per determinare la velocità richiesta per i trasferimenti di I/O, la velocità di elaborazione delle istruzioni, ed il tempo di risposta alle interruzioni.[21] Questi valori, a loro volta, definiscono il minimo numero di operazioni al secondo richieste per il bus, la CPU e i moduli di memoria. I criteri di temporizzazione del sistema sono in parte determinati dai requisiti di I/O del computer, come mostrato nella Tab. 14.9. Poiché i dati per l'ingresso o per l'uscita sono generalmente contenuti in un'area di buffer della memoria durante il trasferimento, è necessario determinare la dimensione in byte dell'area di buffer, come pure le velocità globali dei dati per tutti i canali di I/O. L'area di buffer può essere situata sul modulo stesso, oppure i dati possono essere memorizzati in un'area di buffer della memoria principale.

Inoltre, devono essere definiti i requisiti di memorizzazione nella memoria primaria e della capacità di memorizzazione su disco. Una specificazione esatta potrebbe essere difficile, ma una stima può essere effettuata, eventualmente basata sull'esperienza acquisita del progettista con sistemi simili. Anche il numero ed il tipo di unità periferiche devono essere definiti per il sistema. Le loro caratteristiche determinano il tipo di moduli di controllore da includere nel sistema. Le loro velocità di trasferimento di dati formano una parte delle specifiche dei requisiti di I/O.

Altri fattori nella progettazione del sistema. Anche se i moduli di VMEbus di qualsiasi produttore dovrebbero soddisfare le specifiche del VMEbus, tali moduli variano per il costo, la qualità e l'affidabilità. Prima della selezione, un progettista dovrebbe prendere in considerazione questi fattori, anche se i moduli proposti soddisfano altri criteri del progetto. Altri fattori da considerare possono essere gli intervalli di temperatura e di umidità e l'entità delle vibrazioni che il sistema dovrà sopportare. Il progettista può selezionare moduli adatti per ambienti "ostili", che risulteranno più adeguati dei moduli standard se il sistema dovrà operare a livelli insoliti di temperatura, umidità o vibrazione.

Una decisione importante nella progettazione del sistema è spesso determinata dalla disponibilità di moduli di VMEbus e del software per controllarli. Si tratta di decidere se costruire o acquistare i componenti. In generale, è consigliabile acquistare i moduli di VMEbus e i loro driver di dispositivo per operazioni ordinarie, tranne forse quando il modulo dovrà far parte di un sistema da produrre in grandi quantità. Naturalmente, quando un'applicazione necessita di un modulo speciale, il progettista di sistema non ha altra scelta che quella di progettare, costruire e collaudare il modulo ed il suo driver di dispositivo.

[21] I programmi di *benchmark* sono spesso usati a tal fine. Vari riferimenti bibliografici relativi a questo capitolo, riportati nell'App. E alla fine del libro, discutono il metodo del benchmark per assistere il progettista nel compito di effettuare una stima delle prestazioni del sistema.

14.5.2 Avviamento del sistema

Il processo di progettazione del sistema dovrebbe produrre una configurazione di hardware e di software che soddisferà i requisiti indispensabili di un'applicazione. Un documento di progetto dovrebbe contenere almeno gli elementi descritti di seguito, al fine di descrivere il sistema:

(a) Un diagramma a blocchi che illustri i moduli di VMEbus e le unità di I/O e le loro connessioni.

(b) Una mappa di memoria che mostri gli indirizzi dei vari componenti e qualsiasi partizione della memoria.

(c) Una lista di priorità d'interruzione e di priorità di richieste di bus per i vari elementi del sistema.

(d) Una lista delle componenti software, inclusi i programmi del sistema operativo, il software di sviluppo ed i programmi applicativi.

Una volta che il progetto è stato completato e che i componenti sono disponibili, il progettista di sistema deve mettere insieme fisicamente il sistema stesso e quindi installare il sistema operativo e l'altro software necessario. Questa inizializzazione o "avviamento" del sistema, come descritto qui, si svolge in due fasi. La prima serve a definire le caratteristiche dei singoli moduli di VMEbus, così come operano sul VMEbus. La seconda fase è il processo denominato *generazione del sistema*, che serve a definire il sistema operativo richiesto per lo specifico sistema di computer da creare.

La configurazione dell'hardware. In un sistema di compuer VMEbus con un modulo di CPU ed un controllore di sistema, moduli di memoria e moduli di I/O, è necessario definire le caratteristiche specifiche dell'hardware dei moduli, prima che siano connessi tra loro nel subrack di VMEbus. Il procedimento fondamentale è illustrato nella Fig. 14.12. Le selezioni sono effettuate sui moduli mediante ponticelli o interruttori, seguendo le indicazioni fornite dal manuale per l'utente del modulo. L'assegnazione delle priorità d'interruzione, delle priorità di richiesta di bus, e degli indirizzi determina l'attività globale dell'hardware del sistema.

Le interruzioni che si presentano sul VMEbus sono generalmente classificate in quattro categorie, come segue:

(a) Interruzioni hardware di sistema (p. es., malfunzionamento dell'alimentazione o errore di parità di memoria)

(b) Interruzioni di timer

(c) Interruzioni di I/O

(d) Miscellanee

Selezione delle caratteristiche
dei moduli di VMEbus

↓

Definizione delle priorità d'interruzione,
delle priorità di richiesta di bus
e degli indirizzi del VMEbus

↓

Inserimento del controllore di sistema
nello slot 1 del subrack ed inserimento
degli altri moduli

↓

Connessione delle unità periferiche
e del terminale dell'operatore

↓

Applicazione della potenza AC e test
dei moduli col programma monitor

↓

Pronto per SYSGEN

Fig. 14.12 *Inizializzazione dell'hardware del VMEbus.*

Ogni interruzione ha una priorità sul bus ed interrompe la CPU al livello appropriato. Una possibile selezione di priorità è mostrata nella Fig. 14.13, in cui le interruzioni dovute a malfunzionamenti del sistema hanno la priorità massima (livelli 6 e 7). L'interruzione di timer a livello 5 è per la temporizzazione del sistema globale. Essa non rappresenta un'interruzione dalla circuiteria di temporizzazione di un singolo modulo di CPU che viene usato per scopi di temporizzazione "locale" del modulo. Il lettore deve ricordare che di solito nei sistemi basati sull'MC68020 si utilizza il segnale $\overline{BERR}$, anziché un livello d'interruzione, per indicare una condizione di tempo scaduto (*timeout*) di VMEbus. Tale condizione si presenta quando i trasferimenti di dati tra la CPU e la memoria locale o di VMEbus non vengono completati correttamente, come descritto nel par. 11.5. Naturalmente, niente impedisce ad un progettista di sistema di utilizzare una linea di segnale d'interruzione per indicare un errore di temporizzazione.

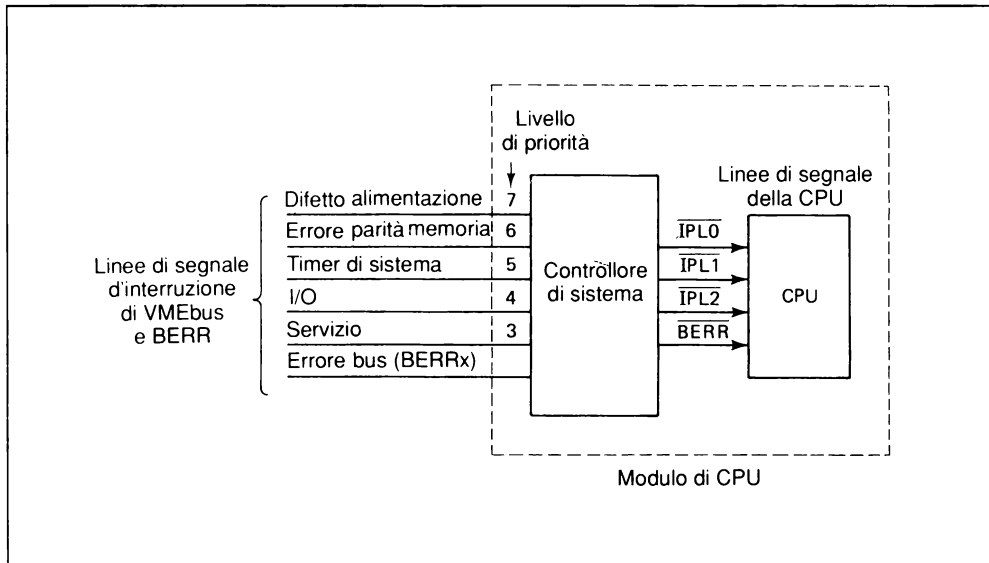



Fig. 14.13 *Esempio di priorità d'interruzione.*

Le interruzioni per i trasferimenti d'ingresso/uscita hanno generalmente una priorità inferiore a quella relativa agli errori dell'hardware o alla temporizzazione del sistema. Si potrebbe scegliere un solo livello, indicato come livello 4 nella figura, per consentire a vari dispositivi di I/O di effettuare interruzioni a questo livello. È necessaria una circuiteria ulteriore per determinare la sottopriorità di questi dispositivi, poiché un solo dispositivo alla volta può interrompere su un singolo livello.[22] Il livello d'interruzione di servizio può essere incluso in alcuni sistemi. Esso indica di solito la situazione in cui un dispositivo di I/O ha completato il trasferimento di dati. Per esempio, un dispositivo che impiega il trasferimento di DMA potrebbe usare il livello d'interruzione di servizio per indicare al sistema operativo che è pronto per un altro trasferimento di un blocco di dati.

Le priorità d'interruzione dei moduli sono fissate in base al progetto del sistema, per cui i dispositivi sono controllati dalle routine di gestione dell'interruzione della CPU. Pertanto, nella selezione delle priorità d'interruzione entrano in gioco anche alcune considerazioni sul progetto del software.[23] L'accesso al bus, comunque, è determinato ciclo per ciclo dalla circuiteria di arbitrato del controllore del sistema di VMEbus, rispettando le priorità dei vari moduli nella richiesta del bus. Una volta stabilite, nessun programma potrà modificare tali priorità di bus, che sono

[22] La selezione delle sottopriorità si basa sulle caratteristiche dei dispositivi che possono causare un'interruzione al medesimo livello. Alcuni dispositivi richiedono un servizio d'interruzione più rapido (sottopriorità superiore) rispetto ad altri.

[23] Per esempio, una routine d'interruzione potrebbe mascherare un'interruzione di livello superiore, poiché la routine nel modo di supervisore può manipolare i bit della mschera d'interruzione dei registro di stato della CPU. Ciò non è comune quando il sistema ha una capacità d'interruzione di priorità.

definite come una parte del progetto di sistema, in base alla velocità di trasferimento e ad altre caratteristiche dei dispositivi che utilizzano il bus. Nella maggior parte dei sistemi con un modulo di CPU ed un controllore di disco, la CPU ha la minima priorità di accesso al bus. Ciò è dovuto al fatto che un trasferimento di dati da o verso un'unità a disco non dovrebbe essere interrotto a questo livello fondamentale dell'hardware. Nalla maggior parte dei sistemi, il controllore di DMA dell'unità a disco "ruba" relativamente pochi cicli di elaborazione della CPU.

Una volta che gli indirizzi per i moduli sono stati stabiliti, il sistema può essere assemblato e collaudato come indicato nella Fig. 14.12. Molti moduli di CPU hanno un programma di monitor in ROM, che consente all'operatore di produrre un auto-test del sistema, come descritto nel par. 5.1 per il modulo MVME133 e per il monitor 133BUG. La procedura dipende dallo specifico programma di monitor per il modulo della CPU. Allorché i componenti dell'hardware funzioneranno correttamente, il sistema operativo ed altri programmi dovranno essere caricati nell'unità a disco del computer.

Generazione del sistema col sistema operativo VERSAdos. Il sistema operativo acquistato del produttore consiste generalmente di un gran numero di routine e di componenti software adatti per una certa varietà di sistemi di computer. Per esempio, la versione completa del sistema operativo VERSAdos della Motorola occupa circa 16 megabyte di memoria, inclusi i suoi programmi per lo sviluppo del software.[24] Il suo nucleo (RMS68K) ha una dimensione di circa 20 kilobyte. Il sistema operativo di base VERSAdos col suo software di gestione dei file occupa circa 128 kilobyte nella memoria. La porzione residente contiene il driver di dispositivo per ciascun modulo di VMEbus e per ogni unità periferica controllata dal VERSAdos. Ovviamente, la maggior parte dei sistemi richiederà soltanto alcuni dei molti componenti software offerti dal sistema operativo VERSAdos completo. Il procedimento per cui un operatore seleziona le porzioni di VERSAdos destinate ad essere residenti nella memoria è noto come *generazione del sistema*, abbreviato in SYSGEN (*SYStem GENeration*).[25] Il risultato di SYSGEN è un sistema operativo modificato, che contiene soltanto le componenti del software richieste per un computer specifico.

Un operatore esegue una generazione del sistema durante una sessione interattiva al terminale dell'operatore. Quando il computer viene inizializzato per la prima volta, le principali componenti del sistema operativo da utilizzare vengono caricate dai dischi flessibili nel disco Winchester nella maggior parte dei sistemi di VMEbus. La versione caricata nella memoria include il nucleo (il *kernel* RMS68K) ed altro software residente già descritto nel par. 14.4.

---

[24] Questa discussione descrive il processo di generazione del sistema per il VERSAdos della Motorola. Altri sistemi operativi impiegano procedimenti simili, che ovviamente differiscono nei dettagli.

[25] Le componenti non usate del software possono essere immagazzinate nell'unità a disco del computer o rimosse dal sistema.

ESERCIZI

14.5.1

Si considerino i requisiti del sistema per la sicurezza dei dati in un ambiente multiutente, allo scopo di proteggere ciò che segue:

(a) L'accesso al sistema (cioè, *log-on*)
(b) Il sistema operativo
(c) I task di utente
(d) Le risorse del sistema quali i file su disco.

Si considerino sia le componenti software che quelle hardware per fornire la sicurezza affinché: gli utenti non autorizzati non possano utilizzare il sistema (1), e sia impedito l'accesso e la modifica alle sue componenti (2, 3 e 4) da parte di programmi che non godono del privilegio necessario.

14.5.2

Si supponga che un trasferimento di DMA sia avviato sul VMEbus. In media, la CPU esegue istruzioni che creano 3 milioni di cicli di bus al secondo. Se il dispositivo di I/O trasferisce una word in un ciclo di bus ogni 10 millisecondi, quanto spesso un ciclo di bus viene "rubato" alla CPU per il trasferimento di DMA (cioè, qual è il rapporto tra cicli di DMA e cicli di CPU)?

14.5.3

Si consideri un modulo di convertitore A/D a quattro canali. Se ciascun canale viene campionato alla frequenza di 40000 campioni di 16 bit al secondo, per 1024 campioni per canale, qual è la specifica del sistema per quanto segue?

(a) Velocità di trasferimento di dati alla memoria, in megabyte al secondo.
(b) La dimensione richiesta del buffer, in byte.
(c) La durata totale di campionamento per campionare tutti i canali al fine di riempire il buffer.
(d) Il rapporto tra il numero di cicli di bus di DMA per trasferire un valore di 16 bit alla memoria ed il numero di cicli di CPU specificati come 3×10^6 cicli di bus al secondo.

14.5.4

Si confronti l'accuratezza prevista (massima e minima) quando le stime delle prestazioni dell'hardware e del software di un sistema sono basate du quanto segue:

(a) La velocità di elaborazione pubblicata dal produttore per la CPU, espressa in milioni di istruzioni al secondo.
(b) Programmi standard di benchmark per la CPU.
(c) Programmi di benchmark eseguiti su una piastra di valutazione, come il modulo MVME133.
(d) Programmi applicativi per un'applicazione particolare o un sistema "prototipo" simile al sistema finale da utilizzare.

14.5.5

Secondo un conteggio effettuato dall'autore, si possono definire 261 parametri durante SYSGEN per il solo modulo MVME133. Considerando sia le componenti sul bus locale (I/O seriale, MC68901, memoria, monitor, ecc.) come definite nel cap. 13 e gli altri moduli presenti nel sistema dell'autore (MVME225-2, MVME320 e unità a disco), il lettore ne elenchi il maggior numero di parametri. Essi sono definiti dai loro valori "normali" (*default*) Nel *VERSAdos to VME Hardware and Software Configuration User's Manual* (MVME VDOS/03) della Motorola, che è uno dei manuali offerti nel pacchetto di documentazione del sistema VERSAdos.

14.6 ALTRI BUS, CPU E SISTEMI OPERATIVI

In questo libro ci si è concentrati sulle caratteristiche e gli impieghi dei computer basati sull'MC68020, con un VMEbus ed il sistema operativo VERSAdos. Come il lettore certamente saprà, esistono molti altri bus, CPU e sistemi operativi largamente diffusi per microcomputer. Essi sono descritti dettagliatamente in vari articoli citati nei riferimenti bibliografici relativi a questo capitolo, elencati nell'App. E alla fine del libro. In questo paragrafo, saranno descritti vari modi per progettare sistemi di computer che impiegano altri bus, CPU o sistemi operativi con componenti della famiglia di prodotti dell'MC68020.

La Fig. 14.14 mostra un certo numero di possibili sistemi in cui sono mescolati alcuni membri della famiglia dell'MC68020 con componenti solitamente non associati alla linea di prodotti della Motorola. Tali componenti sono prodotti hardware e software disponibili per creare un sistema "misto" che non richiede alcun impegno di progettazione, se non quello di selezionare i componenti per soddisfare i requisiti di una particolare applicazione. Per esempio, esistono moduli di CPU che utilizzano CPU della Motorola che si possono connettere direttamente ad altri bus di microcomputer, come mostrato nella Fig. 14.14(a). Il Multibus della Intel, il NuBus della Texas Instrument, ed il PC (*Personal Computer*) dell'IBM non sono

(a) *CPU della Motorola per un altro bus.*

(b) *Un'altra CPU per VMEbus.*

(c) *Modulo da bus a bus.*

(d) *Un altro sistema operativo.*

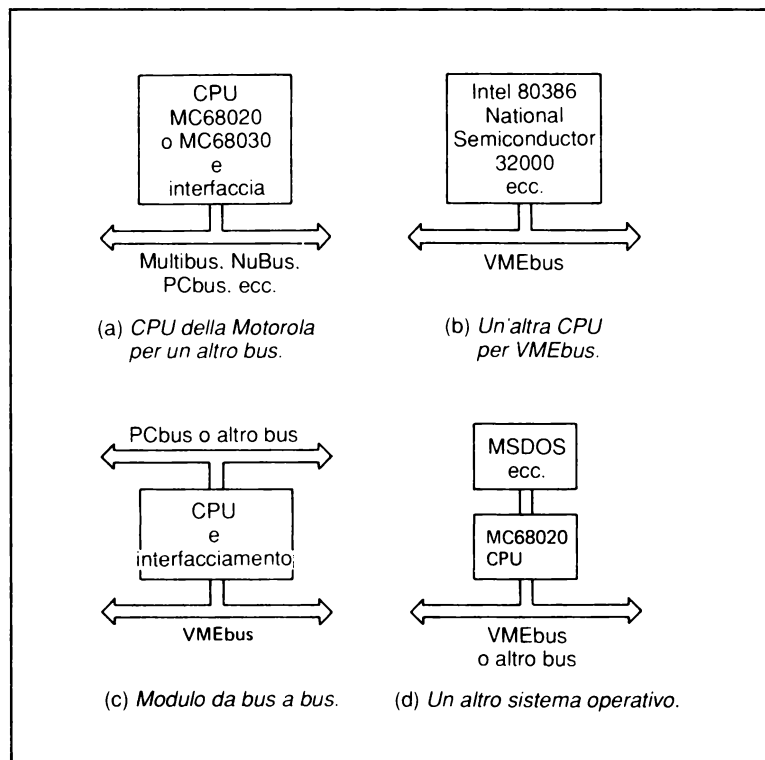



Fig. 14.14 *Componenti dell'MC68020 con altri bus, CPU o sistemi operativi.*

che tre dei possibili esempi di bus che possono ospitare CPU della Motorola quando viene acquistato il modulo appropriato. Un modulo di esempio è la piastra di coprocessore DSI-020 della Definicon. Esso contiene i chip MC68020/MC68881/ MC68851 e si connette al bus del PC.[26] Questa piastra è per coloro che desiderano disporre delle prestazioni di elaborazione dell'MC68020 a 32 bit e del suo coprocessore ma utilizzano un personal computer per lo sviluppo del software.

Come mostrato nella Fig. 14.14(b), si possono connettere altri processori al VMEbus. I processori dell'Intel (80286, 80386, ecc.) e la serie 3200 di CPU della National Semiconductor non sono che due esempi per cui è disponibile una piastra di processore compatibile col VMEbus. Una piastra di questo tipo consente ad un programmatore di utilizzare il software sviluppato per la particolare CPU in questione, ma anche di accedere ai moduli sul VMEbus. Due sistemi di computer possono essere interconnessi come indicato nella Fig. 14.14(c). Solitamente, il software sarà sviluppato su un personal computer, e nel sistema saranno impiegati sia i moduli per il PCbus che quelli per il VMEbus. Quest'ultimo è molto più veloce e consente trasferimenti di 32 bit, mentre il PCbus è limitato a 8 o a 16 bit, a seconda del particolare PCbus utilizzato.[27]

Un altro metodo per lo sviluppo del software è possibile col sistema mostrato nella Fig. 14.14(d). La CPU MC68020 è usata per eseguire i programmi di un sistema operativo normalmente concepito per la serie di processori della Intel. Il sistema operativo MS-DOS (*Microsoft Disk Operating System*) è utilizzato col PC IBM e con numerosi altri personal computer.

ESERCIZI

14.6.1

Si costruisca una tabella per confrontare le caratteristiche di vari bus di microcomputer, quali: STDbus, Multibus I e II, VMEbus, IBM PC, IBM PC/AT (*Advanced Technology*), NuBus, e Future bus. Si confrontino il numero di standard IEEE, le capacità di dati e di indirizzi, le capacità di richiesta d'interruzione e di bus, le caratteristiche fisiche, e la velocità di trasferimento su bus in megabyte al secondo. Oltre a questo libro, è necessario consultare la letteratura tecnica specifica.

14.6.2

Si confrontino vantaggi e svantaggi delle seguenti scelte di sistema:

(a) Acquisto di un sistema VMEbus e del software da un singolo produttore.
(b) Acquisto di moduli e del software da diversi produttori ma basandosi su un sistema con VMEbus e MC68020.
(c) Mescolanza delle linee di prodotti, come mostrato nella Fig. 14.14.

14.6.3

Si descrivano vantaggi e svantaggi della creazione di sistemi come mostrato nella Fig. 14.14. Si consideri ciascuno dei quattro casi mostrati.

---

[26] L'articolo di Wilcox elencato nei riferimenti bibliografici per questo capitolo descrive l'uso della piastra di coprocessore della Definicon e di un PC IBM per eseguire calcoli scientifici.

[27] Il PCbus ha un bus di dati di 8 bit ed un bus di PC/AT che permette trasferimenti di dati di 16 bit.

CAPITOLO 15

IL MICROPROCESSORE MC68030

Nel 1987, la Motorola presentò il nuovo processore MC68030 per offrire una CPU di elevate prestazioni con un'unità di gestione della memoria incorporata nel chip. Come illustra il diagramma a blocchi della Fig. 15.1, l'MC68030 ha un'unità di gestione della memoria (*Memory Management Unit:* MMU) e due memorie cache. Per il resto, questo processore è identico all'MC68020. La nuova CPU fu prodotta per offrire una maggiore velocità di esecuzione delle istruzioni ed un supporto maggiore ai sistemi operativi come UNIX che richiedono unità di gestione della memoria. La Motorola produce anche il coprocessore in virgola mobile MC68882 per sistemi basati sull'MC68020 o sull'MC68030. Il nuovo coprocessore è funzionalmente identico all'MC68881 descritto nel par. 12.3, ma esegue le istruzioni più velocemente.

In questo capitolo saranno presentate le principali differenze tra l'MC68020 e l'MC68030. Poiché il modello di programmazione nel modo di utente dei due processori è identico tranne che per due istruzioni, la programmazione nel modo di utente non sarà discussa qui. Nel par. 15.1 saranno descritte le differenze dei modelli di programmazione nel modo di supervisore, tra cui il controllo del cache di dati e della MMU sul chip. Nel par. 15.2 saranno definite le principali variazioni nel progetto dell'interfaccia che sono necessarie per l'MC68030. Otto linee di segnale sono state aggiunte all'insieme di linee di segnale dell'MC68020: esse consentono all'MC68030 di eseguire trasferimenti asincroni tra la memoria e la CPU, come pure una speciale modalità operativa denominata "*burst*" (raffica) per riempire le memorie cache. Per il resto, i progetti d'interfaccia per l'MC68020 e per l'MC68030 sono praticamente identici.

La Tab. 15.1(a) riassume le principali caratteristiche dell'MC68030 che lo distinguono dall'MC68020. La MMU dell'MC68030 ha capacità che sono essenzialmente un sottoinsieme di quelle dell'unità di memoria impaginata (*Paged Memory Management Unit:* PMMU) MC68851, come mostrato nella Tab. 15.1(b). I termini nella Tab. 15.1 saranno descritti più dettagliatamente nei prossimi paragrafi. Per altri aspetti, la maggior parte delle caratteristiche dell'MC68020 sono state mantenute nell'MC68030.

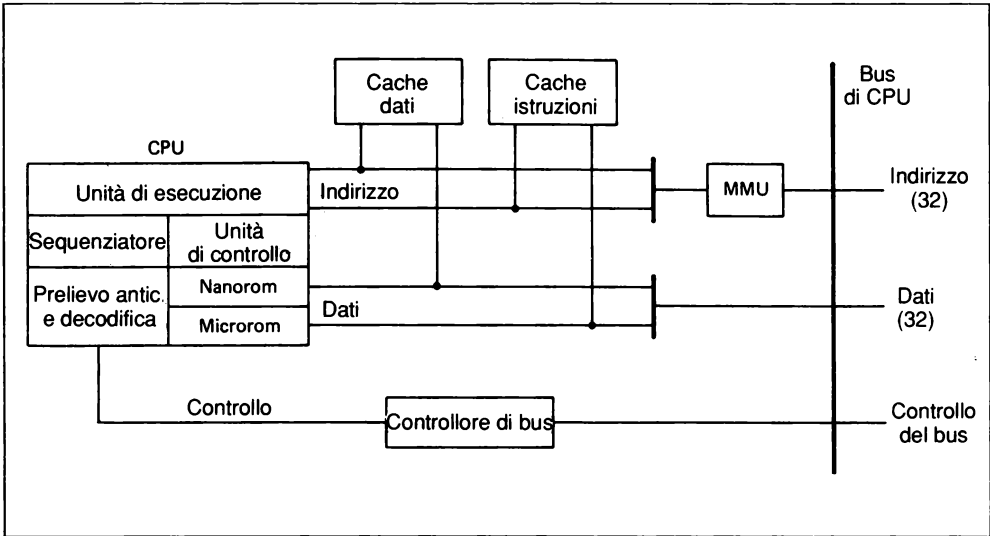



Fig. 15.1 *Diagramma a blocchi semplificato dell'MC68030.*

Altri riferimenti in questo libro. Con poche eccezioni, l'MC68030 è in grado di eseguire le istruzioni dell'MC68020 sia per i programmi in modo di utente che per quelli in modo di supervisore. Altri riferimenti in questo libro per la programmazione dell'MC68030 sono forniti nella Tab. 15.2. Sia il modello di programmazione che l'interfaccia per il coprocessore MC68882 sono identici a quelli dell'MC68881. Gli aspetti di sistema e dell'hardware dell'MC68030 che sono identici a quelli dell'MC68020 sono trattati nei paragrafi indicati nella Tab. 15.2. Il lettore è invitato a rileggere i paragrafi appropriati dei capp. 10, 11, 12 e 13, se lo ritiene necessario, mentre affronta lo studio del materiale presentato in questo capitolo. Nell'app. B è presentato un riepilogo delle differenze tra le famiglie di processori della Motorola.

Tab. 15.1 *(a) Estensioni dell'MC68030 all'MC68020.*

Aspetto	**Caratteristica dell'MC68030**
Modello di programmazione di utente	Identico a quello dell'MC68020, ma l'MC68030 non dispone delle istruzioni CALLM e RTM.
Modello di programmazione di supervisore	Registri supplementari per il controllo della MMU.
Sistema	Cache di dati; MMU su chip.
Hardware	Accesso al cache in un solo ciclo; funzionamento sincrono; riempimento a raffica delle memorie cache.

Tab. 15.1 *(b) Confronto tra MC68020/MC68851 e MC68030.*

Aspetto	MC68020/MC68851	MC68030
Breakpoint (BKPT)	Sì	No
CALLM/RTM	Sì	No
Cache di conversione d'indirizzo	64 entrate	22 entrate
Istruzioni	Istruzioni di controllo della PMMU e istruzioni di coprocessore	Istruzioni di controllo della MMU
Modo trasparente	No	Sì

Tab. 15.2 *Riferimenti all'MC68030 nel testo.*

Aspetto	Riferimento nel testo
Programmazione nel modo di utente	Dal cap. 4 al cap. 9, tranne il par. 4.1 che descrive la CPU MC68020
Programmazione di sistema	Capp. 10 e 11[1]
Interfaccia di coprocessore	Par. 12.2
Coprocessore in virgola mobile MC68882 — modello di programmazione	Par. 12.3 (MC68881)
Istruzioni di multiprocessore (CAS, CAS2, TAS)	Par. 12.6
Chip periferici e programmazione di I/O	Parr. 13.1 e 13.2
Linee di segnale	Par. 13.4[2]
Sistemi di VMEbus	Cap. 14

Note:

1. Esistono differenze minori a causa della presenza della MMU e del cache di dati sul chip dell'MC68030.

2. Le linee di segnale dell'MC68030 per il trasferimento asincrono di dati, i codici di funzione, il controllo delle interruzioni ed il controllo del bus sono simili ma non necessariamente identiche dal punto di vista operativo alle linee di segnale dell'MC68020.

15.1 IL MODELLO DI PROGRAMMAZIONE DI SUPERVISORE DELL'MC68030

La Fig. 15.2 illustra il modello di programmazione di supervisore per il processore MC68030. Il suo insieme di registri è identico a quello dell'MC68020, con l'aggiunta dei registri per controllare la MMU. Tuttavia, il registro di controllo del cache (CACR) ha i bit di controllo per dirigere le operazioni delle memorie cache di istruzioni e di dati dell'MC68030. Le memorie cache saranno considerati all'inizio di questo paragrafo. La MMU su chip sarà trattata nel par. 15.1.2.

31 16 15 0 A7' (ISP) PUNTATORE DI STACK D'INTERRUZIONE
31 16 15 0 A7" (MSP) PUNTATORE DI STACK PRINCIPALE
15 8 7 0 (CCR) SR REGISTRO DI STATO
31 0 VBR REGISTRO DI BASE DEL VETTORE
31 2 0 SFC DFC REGISTRI DI CODICI DI FUNZIONE ALTERNATIVI
31 0 CACR REGISTRI DI CONTROLLO DI CACHE
31 0 CAAR REGISTRI D'INDIRIZZO DI CACHE
63 32 CRP REGISTRO DI PUNTATORE DI RADICE DELLA CPU
63 32 SRP REGISTRO DI PUNTATORE DI RADICE DI SUPERVISORE
31 0 TC REGISTRO DI CONTROLLO DELLA CONVERSIONE
31 0 TT0 REGISTRO 0 DI CONVERS. TRASPARENTE
31 0 TT1 REGISTRO 1 DI CONVERS. TRASPARENTE
15 0 MMUSR REGISTRO DI STATO DI MMU

Fig. 15.2 *Il modello di programmazione di supervisore dell'MC68030. (Per gentile concessione di Motorola, Inc.)*

31 … 14	13					8	7		5	4				0
000000000000000000	WA	DBE	CD	CED	FD	ED	0	0	0	IBE	CI	CEI	FI	EI

WA	= *Write Allocate:*	allocazione di scrittura
DBE	= *Data Burst Enable:*	abilitazione di raffica di dati
CD	= *Clear Data cache:*	cancellazione del cache di dati
CED	= *Clear Entry in Data cache:*	cancellazione di entrata nel cache di dati
FD	= *Freeze Data cache:*	congelamento del cache di dati
ED	= *Enable Data cache:*	abilitazione del cache di dati
IBE	= *Instruction Burst enable:*	abilitazione di raffica di istruzioni
CI	= *Clear Instruction cache:*	cancellazione del cache di istruzioni
CEI	= *Clear Entry in Instruction cache:*	cancellazione di entrata nel cache di istruzioni
FI	= *Freeze Instruction cache:*	congelamento del cache di istruzioni
EI	= *Enable Instruction cache:*	abilitazione del cache di istruzioni

Fig. 15.3 *Il registro di controllo del cache (CACR) dell'MC68030. (Per gentile concessione di Motorola, Inc.)*

15.1.1 I cache di istruzioni e di dati dell'MC68030

Sia il cache di istruzioni che quello di dati dell'MC68030 memorizzano 256 byte disposti come righe su quattro longword. Pertanto ci sono 16 entrate di 16 byte ciascuna.[1] Il cache di istruzioni viene solamente letto dalla CPU ed impiega il codice di funzione FC2 (supervisore o utente) con bit d'indirizzo A3-A31 per selezionare un'entrata. Entrambe le memorie cache sono controllate dal registro di controllo del cache (*CAche Control Register:* CACR) mostrato nella Fig. 15.3. I bit CACR[4:0] controllano il cache di istruzioni, mentre CACR[13:8] controlla il cache di dati. Soltanto il cache di dati è descritto qui poiché il cache di istruzioni opera in maniera pressoché identica a quella dell'MC68020. Tuttavia, un riempimento a raffica di questo cache dell'MC68030 è consentito assegnando il valore {1} al bit CACR[4]. Quando questo bit è posto a {1} e la CPU richiede un'entrata di cache di quattro longword, s'impiega la modalità di riempimento a raffica descritta nel par. 15.2 per aggiornare il cache.

Controllo del cache di dati. Il CACR nella Fig. 15.3 mostra che si possono eseguire le operazioni elencate di seguito per il controllo del cache di dati.

[1] Questa organizzazione è leggermente diversa da quella del formato della memoria cache dell'MC68020 descritta nel par. 12.1.

(a) Abilitazione	CACR[8] = {1}
(b) Congelamento	CACR[9] = {1}
(c) Cancellazione di un'entrata	CACR[10] = {1} ; usando CAAR
(d) Cancellazione del cache	CACR[11] = {1}
(e) Riempimento a raffica	CACR[12] = {1}
(f) Selezione della politica di scrittura	CACR[13]

Naturalmente, se i bit 8-12 sono {0}, l'azione descritta non avrà luogo.[2] Quando la memoria cache è abilitata ed i valori non sono "congelati" per impedirne l'aggiornamento, sono possibili varie altre operazioni. In particolare, CACR[13], il bit di allocazione di scrittura, può essere programmato per determinare l'azione intrapresa dalla CPU quando accede alla memoria o al cache. I vari casi sono riassunti nella Tab. 15.3.

Tab. 15.3 *Operazioni del cache di dati dell'MC68030.*

Operazione	**Modalità operativa**	
	Abilitato/colpo riuscito	**Abilitato/colpo mancato**
Lettura	Lettura dei dati dal cache	Lettura dei dati dalla memoria principale e creazione dell'entrata del cache.
Scrittura		
Sempre	Scrittura dei dati nel cache e nella memoria (scrittura da una parte all'altra)	---
Non allocazione di scrittura	(idem)	Scrittura dei dati nella memoria principale; il cache resta invariato.
Allocazione di scrittura	(idem)	Scrittura dei dati nel cache e nella memoria principale.

Nota: Se il bit di congelamento CACR[9] = {1}, un'entrata non viene sostituita su un "colpo mancato".

[2] Il CACR viene modificato da un programma operante nel modo di supervisore usando l'istruzione MOVEC, come descritto nel par. 12.1.

Un'operazione di lettura è immediata. Se l'entrata viene trovata nella memoria cache quando questa è abilitata su un ciclo di lettura della CPU, allora viene prelevata l'entrata in questione. Ciò rappresenta un "colpo riuscito" (*hit*) nel cache. Altrimenti, per un "colpo mancato" (*miss*), il dato viene trasferito dalla memoria principale alla CPU, e la memoria cache viene aggiornata, a meno che le entrate del cache non siano congelate con CACR[9] = {1}.

La situazione per un'operazione di scrittura nella memoria da parte della CPU è più complessa di quella del ciclo di lettura. Se la locazione della memoria principale in cui scrivere ha un'entrata nella memoria cache, il valore viene scritto sia nel cache che nella memoria principale. Si tratta della cosiddetta politica di scrittura "da una parte all'altra" (*write-through*), che fa sì che l'entrata nel cache ed il valore nella memoria siano sempre identici quando si ha un "colpo riuscito" e la CPU scrive nella memoria.[3] Se così non fosse, il valore nella memoria sarebbe scorretto dopo una scrittura della CPU, supponendo che l'istruzione della CPU dovesse modificare tale valore. Nel caso di un colpo mancato, sono disponibili due opzioni. La scelta viene effettuata tramite un'assegnazione da programma del valore del bit di allocazione di scrittura (*Write Allocate:* WA) CACR[13].

La politica di allocazione di scrittura della CPU determina l'azione intrapresa quando avviene un "colpo mancato" su un ciclo di scrittura della CPU. Nel caso di non allocazione di scrittura con CACR[13] = {0}, una scrittura da parte della CPU modifica la memoria principale soltanto se l'entrata non si trova nella memoria cache. Questa opzione potrebbe essere scelta, ad esempio, in una routine di gestione dell'eccezione di I/O che scrive un valore nel registro di controllo del controllore del dispositivo per avviare un trasferimento di I/O. Il valore non sarà utilizzato di nuovo nella maggior parte dei casi prima che il cache di dati sia riempito da altre istruzioni. Quindi non c'è alcun motivo di trattenere il valore nella memoria cache. Questa selezione di "non allocazione di scrittura" scavalca deliberatamente la memoria cache. Tale politica non sarebbe normalmente utilizzata quando il ciclo di scrittura è volto a modificare un valore nella memoria principale, poiché il cache avrebbe un valore non valido dopo un colpo mancato con una politica di non allocazione di scrittura. La seconda opzione è nota come politica di allocazione di scrittura. Essa serve ad aggiornare sia la memoria cache che quella principale dopo un colpo mancato nel cache.

Una politica di allocazione di scrittura (CACR[13] = {1}) causa un'operazione di scrittura da parte della CPU per aggiornare l'entrata corrispondente nella locazione della memoria principale a cui si accede. La memoria cache contiene quindi il valore più attuale da scrivere, cosicché non esiste alcuna possibilità di errore. La penalità da pagare per questa politica è che in qualche caso le altre entrate in una linea (quattro longword) del cache possono essere invalidate ed è necessario accedere alla memoria per sostituire tali entrate.

[3]Esistono altre politiche per questo scopo; esse sono discusse nei riferimenti bibliografici relativi a questo capitolo, elencati nell'app. E alla fine del libro.

Riempimento a raffica. Ognuna delle due memorie cache può essere caricata con elementi (le "entrate") in due modi:

(a) Entrata singola

(b) Riempimento a raffica

La modalità di entrata singola è selezionata quando CACR[12] = {0}. Un'entrata di longword alla volta viene caricata nella memoria cache quando avviene un colpo mancato su un ciclo di lettura. Questa modalità impiega il trasferimento asincrono dai dati, che richiede almeno tre cicli di clock per ciascun trasferimento. Poiché il cache di dati è riempito ad un'entrata alla volta, sono necessari almeno 12 cicli di clock per aggiornare una linea di cache di quattro longword. Ponendo CACR[12] = {1} per selezionare la modalità a raffica, quattro longword saranno trasferite dopo un colpo mancato su un ciclo di lettura della CPU. Ciò può avvenire in un tempo minimo di cinque cicli di clock, come spiegato nel par. 15.2. L'impiego della modalità a raffica richiede un controllore della memoria principale che può trasferire i dati in questo modo speciale.

15.1.2 L'unità di gestione della memoria dell'MC68030

Il funzionamento e la programmazione della MMU sul chip dell'MC68030 per l'allocazione delle pagine nella memoria è essenzialmente identica a quella dell'unità di gestione della memoria impaginata (*Paged Memory Management Unit:* PMMU) MC68851.[4] La Tab. 15.4 riepiloga le caratteristiche importanti di entrambe le unità di gestione della memoria. Il cache di conversione dell'indirizzo della MMU MC68030 memorizza 22 conversioni da indirizzo logico a fisico, contro le 64 dell'MC68851. La CPU MC68030 ha due registri di puntatore di radice (di CPU e di supervisore) per puntare alle tabelle di pagine nella memoria. Se il sistema operativo trasferisce il controllo ad un task il cui puntatore di radice non è quello corrente, un nuovo puntatore di radice dev'essere prelevato ed inserito nell'appropriato registro del puntatore di radice. Per contro, l'MC68851 memorizza otto puntatori di radice per semplificare il processore di commutazione (di contesto) del task, se il nuovo task ha un'entrata nella tabella di puntatori di radice dell'unità MC68851.

Entrambe le unità di gestione della memoria consentono di designare una pagina come di supervisore o di utente. Tali pagine sono protette dall'accesso della CPU tranne che da parte di un programma operante nella modalità appropriata. Se la protezione viene violata, la MMU causa un'eccezione di CPU.

L'insieme di istruzioni della MMU MC68030 è un sottoinsieme di quello della MC68851 elencato nell'app. C. Come mostrato nella Tab. 15.4, l'MC68030 difetta

[4] La MMU MC68030 non riconosce le istruzioni CALLM o RTM dell'MC68020. La capacità di breakpoint dell'MC68851 dev'essere fornita da una circuiteria esterna se l'istruzione BKPT viene eseguita dall'MC68030. Il chip MC68851 è stato descritto in un certo dettaglio nel par. 12.4.

Tab. 15.4 *Confronto tra l'MC68851 e la MMU dell'MC68030.*

Caratteristica	PMMU 68851	MMU MC68030
Dimensioni della pagina	256 byte — 32 KB	256 byte — 32 KB
Cache di conversione dell'indirizzo	64 entrate	22 entrate
Puntatori di radice alle tabelle	3	2
Tabella di puntatori di radice sul chip (per task separati)	8 entrate	1 entrata
Protezione	Supervisore/utente, protezione di scrittura, livello di accesso (CALLM)	Supervisore/utente, protezione di scrittura
Istruzioni		
Generali	PFLUSH(4), PLOAD(3), PMOVE, PTEST(2), PVALID	PFLUSH, PLOAD, PMOVE, PTEST
Di coprocessore	PSAVE, PRESTORE, PBcc, PDBcc, PScc, PTRAPcc	Nessuna
Conversione trasparente	No	2 segmenti

dell'istruzione PVALID, poiché essa è usata con l'istruzione CALLM della combinazione MC68020/MC68851. Poiché la MMU dell'MC68030 non è un coprocessore, essa non ha le istruzioni di coprocessore, quali PSAVE, PRESTORE, PBcc, PScc e PTRAPcc.[5]

[5] Le istruzioni della MMU MC68030 sono istruzioni di linea F (i bit 12-15 sono {1111}, ma accedono alla MMU sul chip anziché ad un coprocessore esterno. La MMU sul chip ha un identificatore di coprocessore (*CoProcessore IDentifier:* CP-ID) uguale a zero. Quindi sono ammessi sette altri coprocessori in un sistema basato sull'MC68030, in accordo col metodo d'indirizzamento del coprocessore descritto nel par. 12.2.

L'MC68030 ha una caratteristica di conversione "trasparente" che permette l'accesso a segmenti selezionati della memoria senza la conversione degli indirizz. Due registri (TT0 e TT1) nella Fig. 15.2 contengono l'indirizzo dei segmenti trasparenti. La dimensione minima per un segmento è di 16 megabyte.

Eccezioni di MMU. La Tab. 15.5 elenca le eccezioni normalmente associate con l'attività dell MMU. Nella tabella sono definite inoltre le risposte tipiche fornite dal sistema operativo a ciascuna eccezione. Come esempio, un difetto di pagina in un sistema con memoria virtuale potrebbe causare un'eccezione di errore di bus della CPU, che viene servita caricando nella memoria la pagina mancante; poi la CPU prosegue l'esecuzione dell'istruzione cha ha causato il difetto. Altre violazioni che generano un errore di bus dovrebbero causare una fine prematura ("aborto") del programma responsabile.

Tab. 15.5 *Eccezioni dell'MC68030 causate dalla MMU.*

Eccezione	Causa tipica	Risposta tipica
Errore di bus		
Riferimento di memoria non valido	Pagina non presente nella memoria del ciclo	Caricamento della pagina nella memoria e riavviamento
Violazione di privilegio	Errore di programma nel modo di utente	Aborto del programma
Tentativo di scrittura su una pagina protetta contro la scrittura	Errore di programma	Aborto del programma
Eccezione di linea F		
Istruzione scorretta	Incontrata un'istruzione dell'MC68851	Modifica del programma per emulare le istruzioni se il sistema operativo dell'MC68020 è eseguito sull'MC68030
Violazione di privilegio	Un programma nel modo di utente ha tentato di eseguire PFLUSH, PLOAD, PMOVE o PTEST	Aborto del programma
Errore di configurazione (vettore 56)	Errore del sistema operativo	Modifica e ricaricamento del sistema operativo

Se un sistema operativo progettato per un sistema basato sull'MC68020 viene eseguito su un MC68030, si potrebbe presentare un certo numero di eccezioni di linea F. Qualsiasi istruzione per l'MC68851 che non faccia parte dell'insieme di istruzioni dell'MC68020 dovrebbe essere emulata da una routine di eccezione di linea F per mantenere la compatibilità. Qualora un programma nel modo di utente eseguisse una di tali istruzioni si avrebbe una violazione di privilegio tramite l'eccezione di linea F.

Un errore di configurazione si presenta mediante il vettore 56 della tabella di vettori della CPU MC68030, quando il programma di supervisore esegue un'istruzione scorretta per la MMU. Questo è un errore del software di sistema che indica un difetto nel codice del sistema operativo. La tabella di vettori per l'MC68030 è identica a quella dell'MC68020, descritta nel par. 11.1. Tuttavia, i vettori 57 e 58 sono definiti per sistemi di MC68020/MC68851. Questi vettori non sono destinati ad essere usati dalle routine di gestione dell'eccezione dell'MC68030.

ESERCIZI

15.1.1 In quali circostanze si rivela più utile il cache di dati? Cioè, qual è il tipo di problemi risolubili da un programma che trarrebbe il massimo vantaggio dall'impiego del cache di dati?

15.1.2 Si scriva la sequenza di istruzioni per abilitare entrambi i cache e per abilitare il modo di riempimento a raffica.

15.1.3 L'MC68030 impiega due distinti concetti di "architettura". La memoria principale memorizza i dati e le istruzioni insieme, utilizzando il cosiddetto "concetto di von Neumann". Sul chip, bus distinti accedono ai cache di dati e di istruzioni in un'architettura di tipo Harvard. Si discutano i vantaggi e gli svantaggi di ciascuna struttura.

15.2 REQUISITI D'INTERFACCIAMENTO DELL'MC68030

L'MC68030 differisce considerevolmente dall'MC68020 per certi aspetti dei suoi requisiti d'interfacciamento. Pertanto non è possibile sostituire direttamente un MC68020 con un MC68030 in un sistema, al fine di accrescerne le prestazioni. Fisicamente i chip non sono identici e le designazioni dei piedini non corrispondono. Tuttavia, le CPU sono piuttosto simili dal punto di vista funzionale, tranne che per la capacità in più dell'MC68030 di consentire il controllo della memoria cache sul chip e di effettuare trasferimenti sincroni e nel modo a raffica.[6]

[6] L'MC68030 può essere utilizzato per sostituire una CPU MC68020 in un sistema se viene utilizzata una piastra di adattatore per far sì che le varie linee di segnale di ciascun chip corrispondano esattamente. Tale applicazione è descritta nell'*MC68030 User's Manual* disponibile dalla Motorola, Inc.

In questo paragrafo sono presentate le somiglianze e le differenze nell'attività delle linee di segnale per le due CPU. Queste informazioni dovrebbero essere confrontate con la descrizione delle linee di segnale dell'MC68020 del par. 13.4 per comprendere a fondo il funzionamento dell'MC68030.

15.2.1 Le linee di segnale dell'MC68030

La Fig. 15.4 definisce le linee di segnale dell'MC68030, descritte più dettagliatamente nella Tab. 15.6. Per un confronto, la Tab. 15.7 elenca i gruppi funzionali di linee di segnale sia per l'MC68020 che per l'MC68030. Si noti che le linee di segnale per i trasferimenti sincroni di dati, per il controllo delle interruzioni e per il

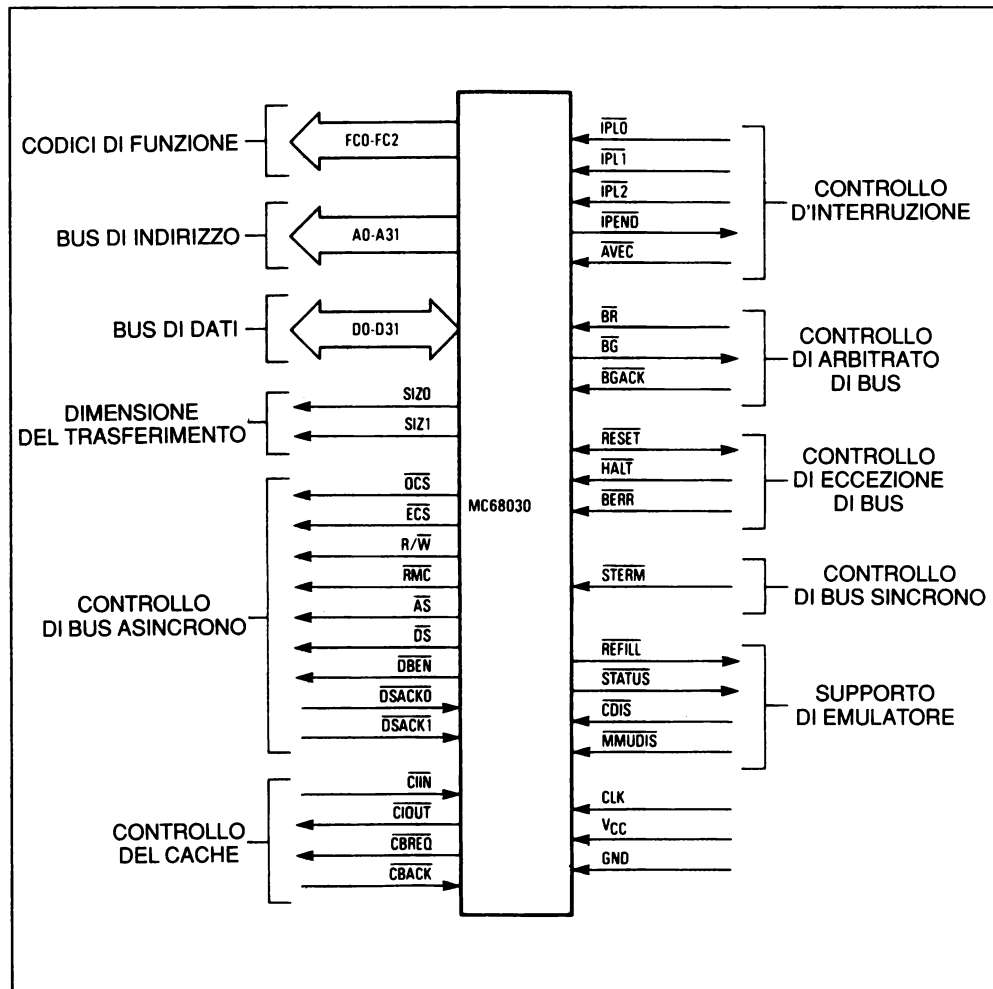



Fig. 15.4 *Diagramma delle linee di segnale dell'MC68030. (Per gentile concessione di Motorola, Inc.)*

controllo dell'arbitrato di bus sono identiche nei due processori. Seguirà una discussione sulla linea di segnale $\overline{\text{HALT}}$ e le linee dei segnali di controllo del cache dell'MC68030. I trasferimenti sincroni ed il riempimento a raffica saranno spiegati nel sottopar. 15.2.2.

Tab. 15.6 *Linee di segnale dell'MC68030.*

Funzione del segnale	**Nome del segnale**	**Funzione**
Codici di funzione (*Function Codes*)	FC0-FC2	Codice di funzione di 3 bit che serve a identificare lo spazio di indirizzi di ciascun ciclo di bus.
Bus d'indirizzo (*Address bus*)	A0-A31	Bus di indirizzo di 32 bit.
Bus di dati (*Data bus*)	D0-D31	Bus di dati di 32 bit, usato per trasferire 8, 16, 24 o 32 bit di dati per ogni ciclo di bus.
Dimensione (*Size*)	SIZ0-SIZ1	Indica il numero di byte che restano da trasferire per il ciclo corrente. Questi segnali, insieme con A1 e A0, definiscono le sezioni attive dal bus di dati.
Inizio del ciclo di operando (*Operand Cycle Start*)	$\overline{\text{OCS}}$	Operazione identica a quella di $\overline{\text{ECS}}$, tranne che $\overline{\text{OCS}}$ viene attivato solo durante il primo ciclo di bus di un trasferimento di operando.
Inizio del ciclo esterno (*External Cycle Start*)	$\overline{\text{ECS}}$	Fornisce un'indicazione che un ciclo di bus sta iniziando.
Lettura/scrittura (*Read/Write*)	R/$\overline{\text{W}}$	Definisce il trasferimento di bus come una lettura o una scrittura del processore.
Ciclo di lettura-modifica-scrittura (*Read-Modify-write Cycle*)	$\overline{\text{RMC}}$	Fornisce un'indicazione che il ciclo di bus corrente fa parte di una operazione indivisibile di lettura-modifica-scrittura.
Abilitazione d'indirizzo (*Address Strobe*)	$\overline{\text{AS}}$	Indica che un indirizzo valido è presente sul bus.
Abilitazione di dati (*Data Strobe*)	$\overline{\text{DS}}$	Indica che un dato valido dev'essere posto sul bus dati da un dispositivo esterno o che è stato posto sul bus dati dalla CPU MC68030.
Abilitazione del buffer di dati (*Data Buffer ENable*)	$\overline{\text{DBEN}}$	Fornisce un segnale di abilitazione per i buffer di dati esterni.
Riconoscimento del trasferimento di dati e della dimensione (*Data transfer and Size ACKnowledge*)	$\overline{\text{DSACK0}}$/ $\overline{\text{DSACK1}}$	Segnali di risposta di bus che indicano il completamento dell'operazione di trasferimento di dati richiesta; inoltre, questi due segnali indicano la dimensione della porta del bus esterno, ciclo per ciclo, e sono usati per i trasferimenti sincroni.
Terminazione sincrona (*Synchronous TERMination*)	$\overline{\text{STERM}}$	Il segnale di risposta di bus che indica una dimensione di porta di 32 bit e che i dati possono essere acquisiti al successivo fronte di salita del clock.

Tab. 15.6 *Linee di segnale dell'MC68030. (continuazione)*

Funzione del segnale	Nome del segnale	Funzione
Inibizione del cache in ingresso (*Cache Inhibit IN*)	$\overline{CIIN}$	Impedisce il caricamento dei dati nei cache di dati e di istruzioni dell'MC68030.
Inibizione del cache in uscita (*Cache Inhibit OUT*)	$\overline{CIOUT}$	Rispecchia il bit CI nelle entrate di ATC o nei registri TTx; indica che i cache esterni dovrebbero ignorare questi accessi.
Richiesta di raffica di cache (*Cache Burst REQuest*)	$\overline{CBREQ}$	Indica una richiesta di raffica per il cache di istruzioni o di dati.
Riconoscimento di raffica di cache (*Cache Burst ACKnowledge*)	$\overline{CBACK}$	Indica che il dispositivo acceduto può operare nel modo a raffica.
Livello di priorità d'interruzione (*Interrupt Priority Level*)	$\overline{IPL0}$-$\overline{IPL2}$	Fornisce un livello d'interruzione codificato per il processore.
Interruzione in sospeso (*Interrupt PENDing*)	$\overline{IPEND}$	Indica che un'interruzione è in sospeso.
Autovettore (*AutoVECtor*)	$\overline{AVEC}$	Richiede un autovettore durante un ciclo di riconoscimento di un'interruzione.
Richiesta di bus (*Bus Request*)	$\overline{BR}$	indica che un dispositivo esterno richiede la padronanza del bus.
Concessione di bus (*Bus Grant*)	$\overline{BG}$	Indica che un dispositivo esterno può assumere la padronanza del bus.
Riconoscimento di concessione di bus (*Bus Grant ACKnowledge*)	$\overline{BGACK}$	Indica che un dispositivo esterno ha assunto la padronanza del bus.
Reset (*RESET*)	$\overline{RESET}$	Reset del sistema.
Arresto (*HALT*)	$\overline{HALT}$	Indica che il processore dovrebbe sospendere l'attività del bus.
Errore di bus (*Bus ERRor*)	$\overline{BERR}$	Indica che è stata tentata una operazione di bus erronea.
Disabilitazione del cache (*Cache DISable*)	$\overline{CDIS}$	Disabilita dinamicamente il cache sul chip per assistere il supporto di emulatore.
Disabilitazione della MMU (*MMU DISable*)	$\overline{MMUDIS}$	Disabilita dinamicamente il metodo di conversione della MMU.
Rifornimento del pipeline (*pipeline REFILL*)	$\overline{REFILL}$	Indica che l'MC68030 sta iniziando a riempire il pipeline.
Stato del microsequenziatore (*microsequencer STATUS*)	$\overline{STATUS}$	Indica lo stato del micro-sequenziatore.
Clock (*CLocK*)	$\overline{CLK}$	Ingresso di clock al processore.
Alimentazione (*power supply*)	V_{CC}	Alimentazione di potenza elettrica.
Massa .(*GrouND*)	GND	Connessione di massa.

Fonte: Motorola, Inc.

Tab. 15.7 *Linee di segnale dell'MC68020 e dell'MC68030.*

Funzione	MC68020	MC68030
Trasferimento asincrono, codici di funzione e linee di segnali di indirizzo e di dati	Medesima	Medesima[1]
Controllo dell'interruzione e controllo di arbitrato del bus	Medesima	Medesima
Arresto (HALT)	Bidirezionale	Di solo ingresso
Difetto di doppio bus indicato dalla CPU	HALT (uscita)	STATUS (uscita continua)
Controllo del cache	CDIS	CIIN, CIOUT, CBREQ, CBACK, CDIS
Trasferimento sincrono	---	STERM
Speciale[2] (emulatore)	---	MMUDIS, REFILL

Note:

1. Certi trasferimenti richiedono considerazioni speciali.

2. MMUDIS disabilita la MMU, mentre REFILL indica che il pipeline della CPU sta per essere rifornito quando i segnali sono attivi (BASSI). Il segnale di stato (STATUS) indica l'operazione della CPU, in base al numero di cicli di clock per cui permane nello stato BASSO.

La linea di segnale HALT. Questa linea si comporta come un segnale d'ingresso per entrambi i processori, per arrestare l'esecuzione di un'istruzione al successivo ciclo di bus. Essa viene attivata (cioè, posta nelo stato BASSO) come segnale di uscita dall'MC68020 per indicare un difetto di doppio bus per la CPU. Tuttavia, una CPU MC68030 mello stato di arresto manterrà attiva (BASSA) la linea del segnale di stato (STATUS) finché il processore non sarà riavviato. Quindi la linea di segnale HALT dell'MC68030 non è bidirezionale, com'è invece nell'MC68020.

Le linee dei segnali di controllo del cache. Queste linee nell'MC68030 sono usate tipicamente come descritto di seguito.

(a) *Cache Inhibit IN:* $\overline{\mathrm{CIIN}}$ — Impedisce l'aggiornamento delle memorie cache nella lettura. Questa linea di segnale è ignorata in una scrittura.

(b) *Cache Inhibit OUT:* $\overline{\mathrm{CIOUT}}$ — Inibisce la memoria cache esterna in base al bit CI (*Cache Inhibit:* inibizione del cache) della MMU.

(c) *Cache DISable:* $\overline{\mathrm{CDIS}}$ — Disabilita entrambi i cache.

I segnali d'ingresso ($\overline{\mathrm{CIIN}}$ e $\overline{\mathrm{CDIS}}$) devono essere generati da circuiti esterni quando è necessario controllare l'attività dei cache sul chip. La Fig. 15.5 mostra il diagramma di un possibile sistema in cui sono impiegati questi segnali. L'interfaccia di I/O pone BASSA $\overline{\mathrm{CIIN}}$ quando si accede ad essa, cosicché i dati trasferiti da un dispositivo d'ingresso/uscita non sono posti nella memoria cache sul chip. Gli analizzatori esterni o gli emulatori possono disabilitare entrambi i cache attivando (ponendo BASSA) la linea di segnale $\overline{\mathrm{CDIS}}$.

La linea di segnale *Cache Inhibit OUTput* ($\overline{\mathrm{CIOUT}}$) serve ad impedire che una memoria cache esterna memorizzi dei dati che non devono essere contenuti in una

MC68030
MMU
Cache
Istruzione
Dati
Bus di sistema
$\overline{\mathrm{CI\ IN}}$
Indirizzo, dati, controllo
$\overline{\mathrm{CI\ OUT}}$
Cache esterno
Memoria principale
Interfacce di I/O
$\overline{\mathrm{CDIS}}$
Inibisce il cache se la pagina non vi dev'essere memorizzata
Inibisce il cache di dati quando è indirizzato per trasferire dati alla CPU
Analizzat di sistema
Al bus di sistema
Disabilita i cache per i test

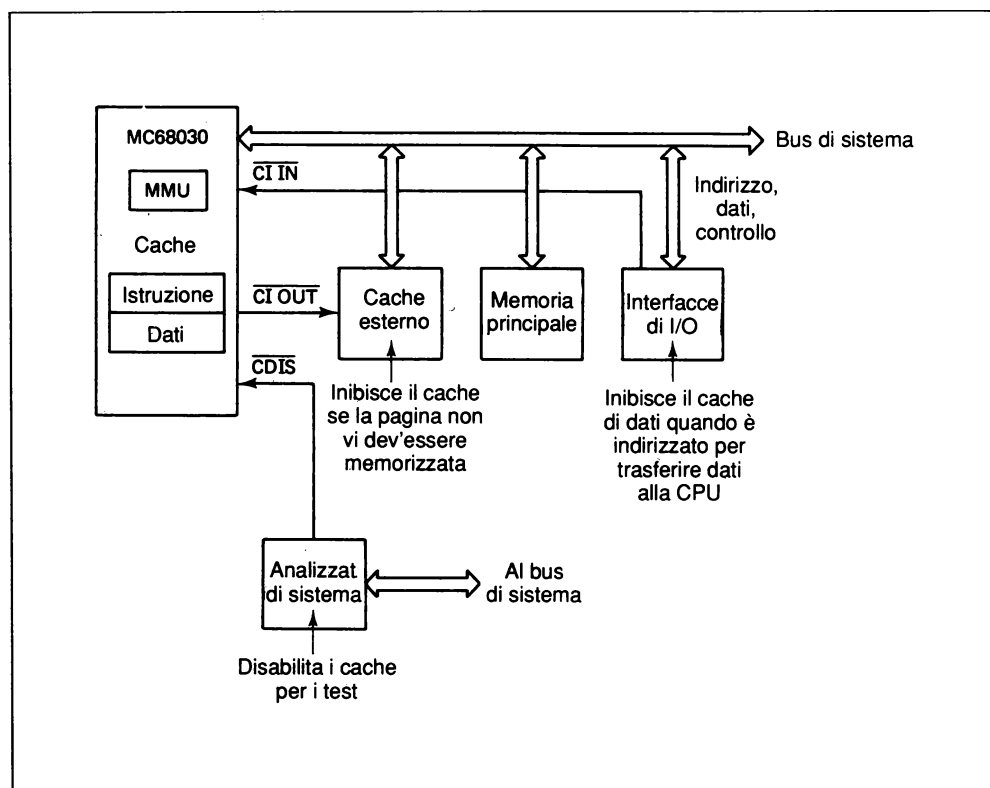



Fig. 15.5 *Connessioni dell'MC68030 e dei moduli esterni, incluso un cache esterno.*

memoria cache.[7] Questo segnale viene attivato (BASSO) quando la MMU indica che una pagina da leggere o scrivere non dovrebbe essere memorizzata nel cache. I cache interni ignorano l'accesso di lettura o di scrittura in questo caso. Anche il cache esterno ignora l'accesso.

15.2.2 Trasferimento sincrono e riempimento a raffica

L'MC68030 può effettuare trasferimenti sincroni e a raffica sul bus di dati. Queste modalità di trasferimento non sono direttamente disponibili col processore MC68020. Il trasferimento *sincrono* è possibile in due cicli di clock anziché in tre, come richiesto per un trasferimento asincrono. Il modo di trasferimento con *riempimento a raffica* può effettuare il caricamento di quattro longword in un cache sul chip in un tempo minimo di cinque cicli di clock. In questo sottoparagrafo sono discussi entrambi i metodi di trasferimento ad alta velocità.

Trasferimento sincrono. Un trasferimento sincrono sul bus di dati della CPU è richiesto da circuiti esterni. La memoria o il dispositivo esterno dev'essere in grado di sincronizzare il trasferimento col clock della CPU. Per esempio, un trasferimento di longword potrebbe avvenire in un tempo di 100 nanosecondi in un sistema con un clock di 20 MHz, poiché ogni ciclo di clock è di 50 ns. Per effettuare tali trasferimenti, il dispositivo esterno deve attivare (porre BASSA) la linea di segnale di terminazione sincrona (*Synchronous TERMination:* $\overline{\text{STERM}}$) durante il primo ciclo di clock di un trasferimento ed acquisire il valore sul bus di dati (ciclo di scrittura della CPU) o presentare il dato sul bus (ciclo di lettura della CPU). Viene indicata la fine del ciclo di trasferimento se il dispositivo disattiva (pone ALTA) $\overline{\text{STERM}}$ entro il ciclo di clock successivo. La dimensione della porta per un dispositivo esterno dev'essere di 32 bit ed i circuiti esterni devono usare il clock della CPU per determinare la temporizzazione esatta per $\overline{\text{STERM}}$. Il segnale di inizio del ciclo esterno (*External Cycle Start:* $\overline{\text{ECS}}$) può essere usato per indicare che la CPU sta iniziando un nuovo ciclo di bus esterno, come spiegato nel par. 13.4. Questo segnale definisce lo stato S0 del clock della CPU.

Riempimento a raffica. La modalità di riempimento a raffica causa un trasferimento sincrono di longword alla CPU usando $\overline{\text{STERM}}$ e le linee di segnale di riempimento a raffica. Se il bit di riempimento a raffica per un cache, contenuto nel registro di controllo del cache, abilita un riempimento a raffica (CACR[4] = {1} o CACR[12] = {1}), la CPU richiede un riempimento a raffica attivando (ponendo BASSA) la linea di segnale di richiesta di raffica di cache (*Cache Burst REQuest:* $\overline{\text{CBREQ}}$) durante il primo ciclo di clock di un accesso esterno alla memoria. Se la memoria risponde attivando (ponendo nello stato BASSO) la linea di segnale di riconoscimento di raffica di cache (*Cache Burst Acknowledge:* $\overline{\text{CBACK}}$) e $\overline{\text{STERM}}$

[7] Una memoria cache esterna viene aggiunta ad un sistema quando le memorie cache sul chip non hanno la capacità di memorizzazione richiesta per soddisfare i requisiti di un'applicazione. Il cache esterno è di solito molto più veloce nel trasferire i valori alla CPU anziché alla memoria principale.

Ciclo di clock 1 2 3 4 5 6 7 8 9 10 11 12

Riempimento a raffica ADR1 LW1 LW2 LW3 LW4 ... trasferimento successivo ...

Trasferimento asincrono ADR1 LW1 ADR1 + 4 LW2 ADR1 + 8 LW3 ADR1 + 12 LW4

Nota: ADR1 è la prima locazione di quattro operandi di longword (LW). Gli indirizzi appaioni in A0-A31. I valori di dati (LW1···LW4) sono presenti in D0-D31 durante i cicli di clock mostrati.

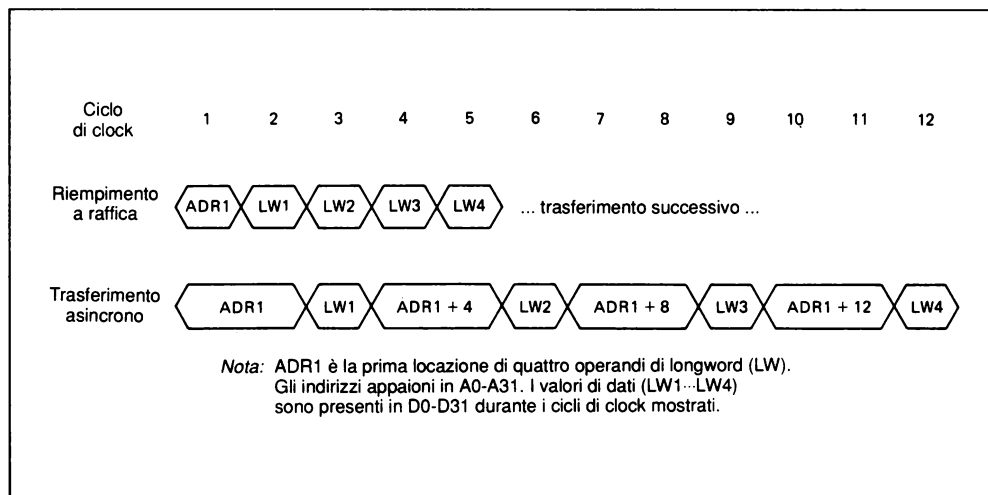



Fig. 15.6 *Confronto tra le operazioni di riempimento a raffica e di trasferimento asincrono.*

durante il trasferimento della prima longword, allora sarà trasferita una seconda longword. Fino a quattro longword possono essere trasferite in questo modo per riempire una linea di cache, come mostrato nella Fig. 15.6. In un riempimento a raffica, le linee d'indirizzo dalla CPU indicano la locazione della prima longword durante l'intero trasferimento. I circuiti di memoria devono fornire gli incrementi agli indirizzi necessari per presentare le altre longword in sequenza, una per ogni ciclo di clock successivo.[8] Pertanto, quattro longword sono trasferite in cinque cicli di clock nel modo a raffica, come mostrato nella Fig. 15.6. Il modo asincrono illustrato nella Fig. 15.6 richiederebbe quattro cicli di lettura completi, cioè dodici cicli di clock.

Riepilogo. L'MC68030 è stato progettato per eseguire le istruzioni dell'MC68020 ad una velocità approssimativamente doppia. Tale aumento della velocità operativa è dovuto in parte all'impiego dei cache di dati e di istruzioni sul chip, che possono essere riempiti dal modo a raffica di trasferimento dalla memoria. Un ulteriore aumentato della velocità si ottiene nei sistemi che richiedono un'unità di gestione della memoria, grazie alla presenza della MMU sul chip. I programmi vengono eseguiti più velocemente, poiché non si perde tempo a convertire indirizzi logici in indirizzi fisici. Al confronto, l'insieme di chip MC68020/MC68851 richiede almeno 45 ns per eseguire ogni conversione d'indirizzo. Infine, l'architettura interna dell'MC68030 consente un alto grado di parallelismo per la maggior parte delle operazioni. Per esempio, l'MC68030 può prelevare un'istruzione e due valori di dati simultaneamente. L'istruzione ed un operando possono essere prelevati dai cache interni, mentre i circuiti di controllo del bus prelevano in maniera indipendente un operando dalla memoria principale.

---

[8] Il progetto dei circuiti di memoria per effettuare i trasferimenti a raffica è descritto nei riferimenti bibliografici relativi a questo capitolo, nell'app. E alla fine del libro. I modelli di memoria che permettono tale modalità sono il modo di pagina, il modo di "nibble" e il modo di colonna statica.

ESERCIZI

15.2.1 Che cosa potrebbe accadere ai prodotti basati sull'MC68020 qualora i produttori impiegassero esclusivamente l'MC68030 nei loro computer di elevate prestazioni?

15.2.2 Quanti piedini utilizza l'MC68030 per le linee di segnale? Ci sono 13 piedini di massa e 10 piedini di V_{CC}.

15.2.3 Si crei una tabella per confrontare le linee di segnale ed altre caratteristiche dei trasferimenti sincroni rispetto a quelli asincroni. Si includa il numero di cicli di clock, la dimensione della porta, il tempo di attivazione dei segnali fondamentali, ed i segnali di terminazione.

15.2.4 L'MC68030 con un ciclo di clock di 25 MHz è talmente veloce nella sua esecuzione che altri dispositivi in un sistema (incluse le memorie ad alta velocità, tranne quelle più veloci) potrebbero non essere in grado di trasferire dati ed istruzioni ad esso alla sua massima velocità. Anche quando ciò è possibile, il bus di sistema è occupato con trasferimenti che riguardano la CPU. In base alle discussioni del cap. 14 e di questo capitolo, si suggeriscano dei progetti alternativi di sistema che potrebbero migliorare le prestazioni globali di un sistema basato sull'MC68030.

RISPOSTE AD ALCUNI ESERCIZI

Capitolo 2

2.1.1 La risposta è reperibile nelle riviste tecniche.

2.1.2 (a) 60 ns
(b) 40 ns

2.1.3 00000-FFFFF; 20 linee di segnale

2.1.4 Altri metodi possono essere adottati per prodotti analogici, per un alto volume di prodotti, o per applicazioni ad altissima velocità.

2.2.1 Byte: 1000, 1001, –1007
Word: 1000, 1002, –1006
Longword: 1000, 1004

2.2.2 (a) 12
(b) 16
(c) 24

2.2.3 (a) 65536
(b) 1048576
(c) 16777216
(d) 4294967296

2.2.4 31 linee di segnale

2.2.6 Alcuni processori a 8 bit effettuano il multiplexing dei segnali nel tempo.

2.3.1 La trappola è causata da alcune condizioni che si presentano nel programma ed è sincrona con l'esecuzione del programma. Un'interruzione è dovuta ad una causa esterna.

2.3.2 I risultati dell'esecuzione di un'istruzione illegale sono imprevedibili, a meno che non sia disponibile una trappola di "istruzione illegale".

2.3.3 Il tempo per R3 è di 20 s + 1.3 s = 21.3 s. L'esecuzione di R2 richiede da un minimo di 31.3 s ad un massimo di 31.3 + 21.3 = 52.6 s. L'esecuzione di R1 richiede da un minimo di 21.3 s ad un massimo di 73.9 s. Il caso peggiore si presenta quando R1 deve attendere per R3 e poi per R2 per poter essere completata.

2.3.4 Le porzioni di un sistema operativo sono eseguite nel modo di supervisore, particolarmente quelle che controllano l'hardware, come le routine di I/O e le routine di gestione delle interruzioni.

2.3.5 Le istruzioni sul chip di solito sono eseguite più velocemente delle istruzioni di coprocessore. L'impiego di coprocessori fornisce più flessibilità al progetto del sistema.

Capitolo 3

3.1.1.1 4.375

3.1.1.2 255.99998

3.1.1.3
(a) 108
(b) 35
(c) 0.975098
(d) 61450

3.1.1.4 x = 5

3.1.1.5 4294967295

3.1.2.1
(a) $1111\ 1001\ 1011\ 1011_2$
(b) $1111\ 1111\ 1111\ 0101_2$
(c) 11010.010_2

3.1.2.2 $100\cdots0_2$

3.1.2.3
(a) Estensione di 0 a sinistra.
(b) Estensione di 1 a sinistra.

3.1.2.4
(a) 346.27
(b) 8223_{16}
(c) 1.1001111

3.1.2.5
(a) –127, +127; –127, +127; –128, +127
(b) –32767, 32767; –32767, 32767; –32768, +32767
(c) –2147483647, +2147483647;
–2147483647, +2147483647;
–2147483647, +2147483647

3.1.2.6 Da –1 a –1000

3.1.3.1
(a) $0100\ 0000\ 0000_2$
(b) $CF08_{16}$
(c) 4294967295
(d) 120_5

3.1.3.2 0.428571428

3.1.3.4 –0.0078125

3.1.3.5 10.00,10.01···11.11,00.00,00.01,···0.11

3.2.1.1
(a) $0000\ 0111_2$
(b) $0001\ 0011_2$
(c) $1001\ 1001_2$

3.2.1.2
(a) 1970
(b) Non valido

3.2.1.3
(a) 99
(b) 9999
(c) 9999 9999

3.2.1.4 Errore se somma ≥ 10*L*

3.2.2.1 128 (BCD)

3.2.2.2 $0001\ 1111\ 1101_2$; moltiplicare la *L*-esima cifra per a0*L* in binario e sommare.

3.2.2.3 Moltiplicare la *L*-esima cifra per il moltiplicatore e poi per 10*L*; convertire il risultato parziale in BCD e sommare nel risultato.

3.2.3.1
(a) $0000\ 0001\ 0010\ 0100_2$
(b) $1001\ 1001\ 1001\ 1001_2$
(c) $1001\ 0000\ 0000\ 0000_2$
(d) Non valido

3.2.3.2
(a) –10; +9
(b) –1000; +999
(c) –10000000; +9999999

3.3.1.2 $0\ 0111\ 1100\ 0000 \cdots_2$

3.3.1.3
(a) $4160\ 0000_{16}$
(b) $\mathrm{BFA0}\ 0000_{16}$

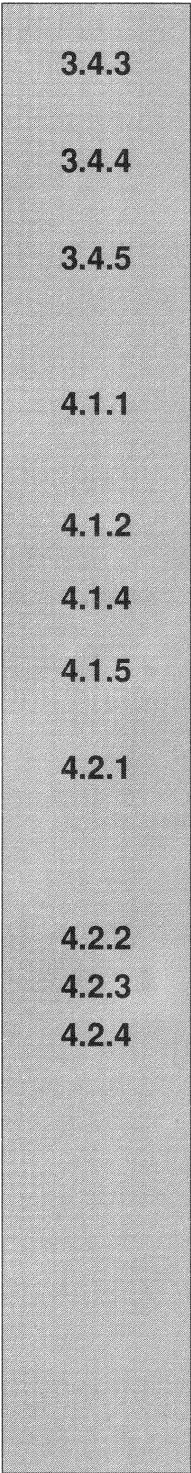
3.3.2.1
(a) $\mathrm{3F00}\ 0000_{16}$
(b) $\mathrm{BF00}\ 0000_{16}$
(c) $0080\ 0000_{16}$

3.3.2.2 2×10^{38} (approssimativamente)

3.3.2.3
(a) $\mathrm{401C}\ 0000 \cdots 00_{16}$
(b) $\mathrm{C03E}\ 0000 \cdots 00_{16}$

3.4.1 500000 byte, cioè il 2.98% della memoria

3.4.2

54 48 45 20	'THE '
4D 4F 54 4F 52 4F 4C 41 20	'MOTOROLA '
4D 43 36 38 30 32 30	'MC68020'

3.4.3

(a) $1111\ 1111_2$
(b) 02 55
(c) 32 35 35

3.4.4

(a) $0010\ 1101_2$
(b) 34 35
(c) 2d 30 34 35

3.4.5

No: il codice Morse è ternario (punto, linea, spazio).

Capitolo 4

4.1.1

In genere, una CPU microcodificata ha un insieme di istruzioni più complesso. La modifica o il colaudo del chip risulta più facile per il produttore. Una CPU progettata con logica "ad hoc" può essere più efficiente (in termini di velocità) per molte applicazioni.

4.1.2

Il processore RISC ha veloci istruzioni di un solo ciclo. Le istruzioni complesse vengono generare in software da un compilatore.

4.1.4

Il pipeline è discusso nei riferimenti bibliografici della Motorola, relativi al cap. 4 di questo libro, elencati nell'app. E.

4.1.5

(a) Miglioramento di 60N nanosecondi ($33^1/_3$%)
(b) Miglioramento di 50N nanosecondi ($33^1/_3$%)

4.2.1

(a) Interruzione di maschera a livello 4
(b) Modo di supervisore
(c) Codice di condizione Z = {1}
(d) Traccia di singola istruzione nel modo di supervisore
(e) Traccia sul cambiamento di flusso; modo di supervisore; interruzioni mascherate.

4.2.2

$(SR)\text{-}2400_{16}$

4.2.3

SR, PC, SSP; USP

4.2.4

Lo stack varia come segue:

Indirizzo (esadecimale)	Interruzione dopo il livello 1	Interruzione dopo il livello 2
007FF0		2100 (SR_1)
007FF2		0000
007FF4		200C (PC_1)
007FF6	libera	formato
007FF8	0000 (SR_0)	0000 (SR_0)
007FFA	0000	0000
007FFC	101C (PC_0)	101C (PC_0)
007FFE	formato	formato

Nell'interruzione di livello 1, (SR) = (2100); nell'interruzione di livello 2, (SR) = (2200). Gli indirizzi e i contenuti sono forniti in esadecimale.

4.3.1

(1000) = 00
(1001) = 00
(1002) = 00
(1003) = 00

4.3.2

(a) (D2)[15:0] = 0801_{16}
(b) (1000) = 1901_{16}
(c) (1000) = 0000
(d) (D2)[15:0] = 0806_{16}
(e) (D1)[15:0] = $1F14_{16}$
(Tutti i valori sono esadecimali nei problemi 4.4.1-4.4.3)

4.4.1

(a) (7D0)[W] = (3E8)
(b) (D1)[W] = (1000)
(c) (D1)[B] = (3E8)
(d) (FF FFFC) = 0000 0000

4.4.2

(a) (D1)[W] = 3E8
(a) (D1)[W] = FFE0
(a) (D1)[W] = FFE0
(a) (D1)[W] = 03E8

4.4.3

(a) (D1)[W] = 4142
(b) (D1)[W] = 00C1
(c) (D1)[W] = 03E8

4.4.4

```
(a) MOVE.W    D1, TEMP
    MOVE.B    TEMP,1001
    MOVE.B    TEMP+1,1002
    (TEMP è l'indirizzo di una locazione di word)
(b) MOVE.W    D1, 1001
```

4.4.5

(a) (D1)[W] = \$1000
(b) (D1)[W] = \$5BFE

4.5.1

```
MOVE.W      VAL1,D0
ADD.W       VAL2,D0
MOVE.W      D0,RISULT
```

4.5.2

(a) (1000) = 00
(b) (1000,1001) = 00 00
(c) (1000) = 1000
(d) (D1)[15:0] = 0010
(e) (D1)[15:0] = 1000
(f) (D1)[7:0] = 10

4.5.3

```
(a) CLR.W     D1
(b) MOVE.L    A3,D0
(c) MOVE.B    #$2E,D0
```

4.5.4

(a) 4240
(b) 2008
(c) DA00
(I valori sono in esadecimale)

4.5.5

Poiché la word di operazione è di 16 bit, sono disponibili 65536 possibilità distinte. L'istruzione MOVE stessa richiede molte migliaia di queste possibili combinazioni di istruzioni.

4.5.6 Il supporto per i sistemi operativi, i linguaggi ad alto livello, ed il software per le workstation d'ingegneria è stato fornito dai progettisti.

4.5.7 In qualche caso, il codice-sorgente per un programma può non essere disponibile.

4.6.1 4294967296 byte; 2147483648 word; 1073741824 longword

4.6.2

(a)	(1000) = 1020	(BCD)
	(1002) = 3040	(BCD)
(b)	(1000) = 0200	
	(1002) = 00FC	
(c)	(1000) = 4142	
	(1002) = 4344	

4.6.3 (1002) = 0002
(1004) = FFF0

(I valori sono esadecimali negli esercizi 2 e 3, a meno che non sia specificato altrimenti).

4.6.4

(1000)xx	12	34	56
(1004)78	xx	xx	xx

Sono richiesti due cicli di scrittura.

Capitolo 5

5.1.1 Il tracciamento di passo singolo equivale ad avere un breakpoint in ogni locazione di istruzione. I breakpoint consentono il debugging di un programma mediante il tracciamento di una sezione di codice anziché di ciascuna istruzione.

5.1.2-5.1.4 Si consiglia di consultare la letteratura tecnica delle case produttrici, le riviste e i libri di testo di informatica e computer.

5.2.1 Programma.

5.2.2

```
        MOVEA.L  #20000,A1
        MOVE.L   #4,D2
LOOP    ADD.W    (A1)+,D1
        SUB.W    UNO,D2
        BNE      LOOP
        JMP      RETURN
UNO     DC.W     1
```

(Anche RETURN dev'essere definita).

5.2.3 (a) 4E 20 49 53 (esadecimale)
(b) 14_{16}
(c) Riserva quattro byte
(d) $AAAA (divisione di interi)
(e) $40000

5.2.5-5.2.8 Varie risposte.

5.3.1 Il PC fa riferimento ad istruzioni che, secondo i progettisti della Motorola, non dovrebbero essere modificate.

5.3.2 Si usi MOVEA.L #0,<An> per azzerare <An>.

5.3.3 Gli indirizzi di operandi definiti dall'indirizzamento assoluto non possono essere modificati dopo l'assemblaggio. Gli indirizzi indiretti e relativi possono essere modificati durante l'esecuzione poiché l'indirizzo effettivo è calcolato mentre l'istruzione viene eseguita.

5.3.4 Gli indirizzi brevi richiedono soltanto una word di estensione ma l'intervallo è limitato a 16 bit.

5.3.5
(a) $0FFF
(b) $1000
(c) $0FFE

5.3.6
(a) ((A0)) ← ((A0) + 4)
(b) (D1)[W] ← ($FFFF9000); indirizzo di sorgente esteso di segno
(c) (D1)[B] ← ((A0) + (D1))

5.3.7 Entrambi i modi estendono di segno il valore a $FFFF8000.

5.3.8
(a) Indirizzo di operando: $10000, $10004, $10008, e così via, in un ciclo.
(c) Sono indirizzati blocchi di 16 longword.

5.3.9
(a) (D3)[W] = $0100
(b) (D4)[W] = $0200

5.3.10-5.3.12 Programmi

5.4.1 Esempio:
ADD.W ($1F00, A1, D1.L*2), D2; (D2)[W] = ($31F00)

5.4.2 Esempio:
ADD.W ([$1F00, A1, D1.L*2], $1000), D2; (D2)[W] = ($1900)

5.4.3 ($01234567) ← $0000

5.4.4 Il codice del linguaggio-macchina non è identico poiché lo spostamento è diverso in ciascuna istruzione. Se fosse usato A1, il codice sarebbe il medesimo.

Capitolo 6

6.1.1 I bit C e V del codice di condizione dovrebbero essere azzerati dopo un trasferimento di dati, affinché i test condizionali operino correttamente.

6.1.2-6.1.3 Programmi

6.1.4 L'impiego di MOVEM risulta in un programma più breve se vengono salvati più registri.

6.1.5 SWAP consente di accedere al byte più significativo in una word. È necessaria un'istruzione di rotazione per accedere singolarmente a tutti i byte in una longword.

6.2.1 La tabella di salto (contenuta nella memoria di lettura/scrittura) consente di modificare gli indirizzi di entrata nella ROM.

6.2.2 (a) (PC) ← ((A2))
(b) A1) ← ((A2))
(c) (D2)[W] ← ((D1))[W]
(d) (PC) ← (((A2)))

6.2.3 (a) BEQ
(b) BGT
(c) BLT

6.2.4-6.2.6 Programmi

6.2.7 BGE, BLT, BGT e BLE effettuano correttamente i test dei risutlati quando V = {0}. BPL e BLT causano un salto quando N = {0}; BLT e BMI causano un salto quando N = {1}, quando V = {0}.

6.2.8 Programma

6.3.1 L'offset è $000E_{16}$. L'istruzione è $61oE_{16}$.

6.3.3 L'annidamento è limitato dalla lunghezza dello stack.

6.3.4 Programma

Capitolo 7

7.1.1 Dopo le operazioni aritmetiche, i bit V o C dovrebbero essere esaminati per l'overflow o il riporto. Dopo MOVE, CMP e TST, V = {0} e C = {0}.

7.1.2 Se V = {1}, BGE produce un salto se N = {1} ma BPL produce un salto se N = {0}.

7.1.3 Il risultato rappresenta −16384; V = {1} indica un errore.

7.1.4 $N_1 + (r^m - N_2) = r^m + N_2 - N_1$). Ciò rappresenta $(N_2 - N_1)$ nell'aritmetica in modulo r^m.

7.2.1 (a) $FF00; C = {1}, V = {0}, Z = {1}
(c) $FFFE; N = {1}
(e) $0000; Z = {1}

7.2.2-7.2.3 Dimostrazione matematica.

7.2.4-7.2.5 Programmi

7.3.1 (a) $0002, $0000
(c) $FFFE, $0000
(d) $FFFE, $FFFF

7.3.2-7.3.3 Dimostrazione matematica

7.3.4 Programma

7.3.5 (a) \$FFFFFFFC; N = {1}, V = {1}
(b) \$FFFFFFFC; N = {1}, V = {1}

7.3.6 (a) \$00000000
(c) \$FFFFFFFF

7.4.1 Interi da 0 a 1.85×10^{19}; frazioni da 0 a 5.4×10^{-20}; interi con segno da -9.22×10^{18} a 9.22×10^{18}.

7.4.2 Programma

7.4.3 (D0) = \$00000001; (D1) = \$FFFFFFFC

7.4.4 (a) V = {1}
(b) (D0) = \$00000000; (D1) = \$FFFFFFFF

7.4.5 Programma

7.5.1 (a) \$00 37
(b) \$99 63

7.5.2 (a) 0435
(b) 2845, X = {1}

7.5.3 426

7.5.4-7.5.5 Programmi

7.6.1-7.6.7 Programmi

Capitolo 8

8.1.1-8.1.3 Programmi

8.2.1 Sei bit sono usati per specificare il conteggio di scorrimento.

8.2.3 +40 con V = {1}

8.3.1 (a) Il bit 0 = {0}
(b) Il bit *m* = {1}

8.3.2 (a) BCLR #15,D2
(b) BSET #15,D2

8.3.5 Un'interruzione che si presenti tra le istruzioni TST e BNE potrebbe consentire l'esecuzione di una routine che esamina il flag prima che sia modificato dal programma interrotto. Quindi viene fornita una falsa indicazione alla routine d'interruzione.

8.4.1 (a) BFCHG inverte i bit in un campo di bit; BCHG inverte un solo bit in un operando di byte, nella memoria, o un operando di longword in un registro di dati; NOT complementa un operando, di byte, di word o di longword. L'istruzione BFCHG imposta i codici di condizione in base al valore iniziale del campo di bit; BCHG fissa Z in accordo al bit da modificare; NOT imposta N e Z per indicare i risultati come negativo o zero, rispettivamente.

8.4.2 (a) N = {0}, Z = {1}
(d) (D3) = $00003037
(g) (D1) = $0000000C

8.4.3-8.4.6 Programmi

Capitolo 9

9.1.1 (a) $3000
(b) $0000 A000

9.1.3 $FFFF A000

9.1.4 (a) (A1) = $11111111
(b) (A2) = $20200
(c) (A1) = $20100

9.2.1 Si usi MOVE.W #1,D1 per inizializzare (D1). Inoltre, "RISULT" come destinazione non può essere un indirizzo relativo.

9.2.2 Il puntatore (A0) è modificato ogni volta che il programma viene caricato ed eseguito.

9.3.6 Poiché $FFF0 < (D3)[W] < $0040, il valore in D3 è nell'intervallo; Z = {0} e C = {0}.

9.4.2 Nessun indirizzo dev'essere passato alla subroutine poiché (PC) è l'indirizzo di base dell'area di parametri in linea. I valori sono fissi, tuttavia.

9.4.5 La sequenza di istruzioni ottiene i medesimi risultati dell'istruzione LINK.

Capitolo 10

10.1.1-10.1.4 Si veda il testo per le discussioni.

10.2.1 (a) Tracciamento disattivato; modo di utente; disabilitazione del livello 4 e delle interruzioni di livello inferiore.
(d) Cambio nel modo di utente.

10.2.2 (a) C = {0}; V = {0}; X = {0}; N e Z attivati.
(c) Trappola (violazione di privilegio)

10.3.1 La sequenza mostrata pone la CPU nel modo di utente prima che sia eseguita l'istruzione JMP. Questa istruzione è nello spazio di memoria di supervisore.

Capitolo 11

11.4.3

Potrebbe presentarsi un'istruzione illegale se un programma non si è concluso correttamente oppure ha causato un trasferimento di controllo ad un'area di memoria contenente valori di dati. Un malfunzionamento della memoria potrebbe causare il prelievo di istruzioni illegali.

11.6.1

Un'interruzione non mascherabile potrebbe essere impiegata per terminare prematuramente ("abortire") un programma che si è bloccato in una situazione di ciclo infinito. Ad esempio, questa interruzione potrebbe essere generata pigiando il pulsante "ABORT" sul modulo MVME133 della Motorola. L'interruzione potrebbe essere altresì utilizzata per avviare una sequenza di "mancanza di alimentazione".

Capitolo 12

12.1.1

Nei casi descritti nel problema, la memoria cache potrebbe non fornire alcun vantaggio nella velocità di esecuzione di un programma. Ciò è dovuto al fatto che le istruzioni nel cache potrebbero non essere valide dopo che un programma ha effettuato un salto o è stato interrotto. Per l'analisi dell'hardware, il cache viene solitamente disabilitato per consentire alla CPU di accedere alla memoria principale per prelevare le sue istruzioni.

12.2.1

(a) $2200A
(b) $22000
(c) $22010

12.3.1

(a) $3F800000

12.4.2

(a) $745A7123
(c) 131072 byte

Capitolo 13

13.1.1

L'I/O rappresentato nella memoria consente a tutte le istruzioni
di far riferimento ad operazioni di I/O e non c'è alcun limite nel numero di porte di I/O entro la capacità d'indirizzamento della CPU. Tuttavia, lo spazio di memoria è usato per l'I/O. L'I/O isolato è solitamente più veloce.

13.2.2

(a) Interruzione
(b) Interruzione o DMA
(c) DMA

13.3.2

(90 + 33 cicli) x 50 ns(ciclo = 61.5 microsecondi

13.4.2

(a) Dati di utente
(b) Programma di utente
(c) Programma di supervisore, ma il riferimento alla destinazione non è lecito.

13.4.3

Il vettore 64 è alla locazione $100. I numeri di vettore da 2 a 15 sono risèrvati per varie eccezioni. Se questi numeri fossero usati per interruzioni, potrebbe sorgere un conflitto se avvenisse un'eccezione col medesimo numero.

Capitolo 14

14.4.2

40 MB/6.55 KB = 61 dischi.

14.5.2

Un ciclo ogni 30000.

14.5.3

(a) 40000 campioni/sec x 4 x 2 byte/word = 0.32 MB/sec
(c) 1024 word a 40000 campioni/sec = 25.6 millisecondi.

Capitolo 15

15.1.2

```
MOVE.L    #$1111,D0
MOVE.L    D0,CACR
```

15.2.2
122 piedini

APPENDICI

APPENDICE A

L'INSIEME DI CARATTERI ASCII E LE POTENZE DI 2 E DI 16

Quest'appendice contiene il codice americano standard per lo scambio di informazioni (*American Standard Code for Information Interchange:* ASCII) ed una tabella di potenze di 2 e di 16. I valori ASCII esadecimali sono utilizzati per rappresentare lettere, numeri ed altri caratteri speciali nella memoria. Nella tabella delle potenze, sono forniti i valori di 2^n e di 16^k, per $n = 0, 1, ..., 32$ e $k = 0, 1, ..., 8$.

Tab. A.1 *L'insieme di caratteri ASCII.*

Carattere	Commenti	Valore hex
NUL	Nullo o avanzamento nastro	00
SOH	Start Of Heading	01
STX	Start Of Text	02
ETX	End Of Text	03
EOT	End Of Transmission	04
ENQ	Enquire (*who are you*, WRU)	05
ACK	Acknowledge	06
BEL	Bell	07
BS	Backspace	08
HT	Horizontal Tab	09
LF	Line Feed	0A
VT	Vertical Tab	0B
FF	Form Feed	0C
CR	Carriage Return	0D
SO	Shif Out (a nastro rosso)	0E
SI	Shif In (a nastro nero)	0F
DLE	Data Link Escape	10
DC1	Device Control 1	11
DC2	Device Control 2	12
DC3	Device Control 3	13
DC4	Device Control 4	14
NAK	Negative Acknowledge	15
SYN	Synchronous Idle	16
ETB	End of Transmission Block	17
CAN	Cancel	18
EM	End of Medium	19
SUB	Substitute	1A
ESC	Escape (prefisso)	1B
FS	File Separator	1C
GS	Group Separator	1D
RS	Record Separator	1E
US	Unit Separator	1F
SP	Space of Blank	20

Tab. A.1 *(continuazione)*

Carattere	Commenti	Valore hex
!	Punto esclamativo	21
"	Virgolette	22
#	Segno di numero	23
$	Segno di dollaro	24
%	Segno di percentuale	25
&	”E” commerciale	26
’	Apostrofo (accento acuto)	27
(	Parentesi tonda aperta	28
)	Parentesi tonda chiusa	29
*	Asterisco	2A
+	Segno più	2B
,	Virgola	2C
–	Segno meno	2D
.	Punto	2E
/	Sbarretta	2F
0	Cifra 0	30
1	Cifra 1	31
2	Cifra 2	32
3	Cifra 3	33
4	Cifra 4	34
5	Cifra 5	35
6	Cifra 6	36
7	Cifra 7	37
8	Cifra 8	38
9	Cifra 9	39
:	Due punti	3A
;	Punto e virgola	3B
<	Minore di	3C
=	Uguale	3D
>	Maggiore di	3E
?	Punto interrogativo	3F
@	”A” commerciale	40

Tab. A.1 *(continuazione)*

Carattere	Commenti	Valore hex
A	Lettera maiuscola A	41
B	Lettera maiuscola B	42
C	Lettera maiuscola C	43
D	Lettera maiuscola D	44
E	Lettera maiuscola E	45
F	Lettera maiuscola F	46
G	Lettera maiuscola G	47
H	Lettera maiuscola H	48
I	Lettera maiuscola I	49
J	Lettera maiuscola J	4A
K	Lettera maiuscola K	4B
L	Lettera maiuscola L	4C
M	Lettera maiuscola M	4D
N	Lettera maiuscola N	4E
O	Lettera maiuscola O	4F
P	Lettera maiuscola P	50
Q	Lettera maiuscola Q	51
R	Lettera maiuscola R	52
S	Lettera maiuscola S	53
T	Lettera maiuscola T	54
U	Lettera maiuscola U	55
V	Lettera maiuscola V	56
W	Lettera maiuscola W	57
X	Lettera maiuscola X	58
Y	Lettera maiuscola Y	59
Z	Lettera maiuscola Z	5A
[	Parentesi quadra aperta	5B
\	Sbarretta.retroversa	5C
]	Parentesi quadra chiusa	5D
^	Accento circonflesso	5E
_	Trattino di sottolineatura	5F

Tab. A.1 *(continuazione)*

Carattere	Commenti	Valore hex
‘	Accento grave	60
a	Lettera minuscola a	61
b	Lettera minuscola b	62
c	Lettera minuscola c	63
d	Lettera minuscola d	64
e	Lettera minuscola e	65
f	Lettera minuscola f	66
g	Lettera minuscola g	67
h	Lettera minuscola h	68
i	Lettera minuscola i	69
j	Lettera minuscola j	6A
k	Lettera minuscola k	6B
l	Lettera minuscola l	6C
m	Lettera minuscola m	6D
n	Lettera minuscola n	6E
o	Lettera minuscola o	6F
p	Lettera minuscola p	70
q	Lettera minuscola q	71
r	Lettera minuscola r	72
s	Lettera minuscola s	73
t	Lettera minuscola t	74
u	Lettera minuscola u	75
v	Lettera minuscola v	76
w	Lettera minuscola w	77
x	Lettera minuscola x	78
y	Lettera minuscola y	79
z	Lettera minuscola z	7A
{	Parentesi graffa aperta	7B
|	Sbarretta verticale	7C
}	Parentesi graffa chiusa	7D
~	Tilde	7E
	Cancellazione (*delete*)	7F

Tab. A.2 *Potenze di 2 e di 16.*

16^k 2^n	n	k	2^{-n}
1	0	0	1.0
2	1		0.5
4	2		0.25
8	3		0.125
16	4	1	0.062 5
32	5		0.031 25
64	6		0.015 625
128	7		0.007 812 5
256	8	2	0.003 906 25
512	9		0.001 953 125
1 024	10		0.000 976 562 5
2 048	11		0.000 488 281 25
4 096	12	3	0.000 244 140 625
8 192	13		0.000 122 070 312 5
16 384	14		0.000 061 035 156 25
32 768	15		0.000 030 517 578 125
65 536	16	4	0.000 015 258 789 062 5
131 072	17		0.000 007 629 394 531 25
262 144	18		0.000 003 814 697 265 625
524 288	19		0.000 001 907 348 632 812 5
1 048 576	20	5	0.000 000 953 674 316 406 25
2 097 152	21		0.000 000 476 837 158 203 125
4 194 304	22		0.000 000 238 418 579 101 562 5
8 388 608	23		0.000 000 119 209 289 550 781 25
16 777 216	24	6	0.000 000 059 604 664 775 390 625
33 554 432	25		0.000 000 029 802 322 387 695 312 5
67 108 864	26		0.000 000 014 901 161 193 847 656 25
134 217 728	27		0.000 000 007 450 580 596 923 828 125
268 435 456	28	7	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29		0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30		0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31		0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	8	0.000 000 000 232 830 643 653 869 628 906 25

APPENDICE B

CONFRONTO DEI MEMBRI DELLA FAMIGLIA DELL'MC68000

Quest'appendice presenta un riepilogo, per consentire un confronto dei processori MC68000, MC68008, MC68010, MC68020 e MC68030. Sono elencate le caratteristiche importanti dei processori per evidenziare le differenze tra le unità centrali di elaborazione (CPU) della famiglia M68000 della Motorola. Tali processori sono stati presentati nel cap. 1 del libro.

Per informazioni più dettagliate in merito alle differenze tra l'MC68000 e l'MC68010, si può consultare l'*M68000 Programmer's Reference Manual.*

	MC68000	MC68008	MC68010	MC68020	MC68030
Dimensione del bus di dati (bit)	16	8	16	8, 16, 32	8, 16, 32
Dimensione del bus di indirizzo (bit)	24	20	24	32	32
Cache di istruzioni (in word)	--	--	3[1]	128	128
Cache di dati (in word)	--	--	--	--	128

Nota 1: L'MC68010 dispone di un cache di 3 word per il modo di ciclo.

Memoria/Macchina virtuale

MC68010, MC68020, MC68030	Rivelazione di errore di bus, recupero dal difetto
MC68030	MMU sul chip

Interfaccia di coprocessore

MC68000, MC68008, MC68010	Emulato in software
MC68020, MC68030	In microcodice

Allineamento di dati di word/longword

MC68000, MC68008, MC68010	I dati di word/longword, le istruzioni e lo stack devono essere allineati alla word.
MC68020, MC68030	Solamente le istruzioni devono essere allineate alla word.

Registri di controllo

MC68000, MC68008	Nessuno
MC68010	SFC, DFC, VBR
MC68020	SFC, DFC, VBR, CACR, CAAR
MC68030	SFC, DFC, VBR, CACR, CAAR, CRP, SRP, TC, TT0, PSR

Puntatori di stack

MC68000, MC68008, MC68010	USP, SSP
MC68020, MC68030	USP, SSP (MSP, ISP)

Bit del registro di stato

MC68000, MC68008, MC68010	T, S, I0/I1/I2, X/N/Z/V/C
MC68020, MC68030	T0/T1, S, M, I0/I1/I2, X/N/Z/V/C

Codice di funzione/Spazio d'indirizzi

MC68000, MC68008	FC0-FC2 = 7 se interruzione riconosciuta, soltanto.
MC68010, MC68020, MC68030	FC0-FC2 = 7 se spazio di CPU

Cicli di bus indivisibili

MC68000, MC68008, MC68010	Impiego del segnale $\overline{AS}$
MC68020, MC68030	Impiego del segnale $\overline{RMC}$

Frame di stack

MC68000, MC68008	Gestiscono l'insieme originale
MC68010	Gestisce i formati $0, $8
MC68020, MC68030	Gestisce i formati $0, $1, $2, $9, $A, $B

Modi d'indirizzamento

Estensioni di MC68020, MC68030	Modi d'indirizzamento indiretto della memoria, indice scalato, e grandi spostamenti. Si consultino le specifiche tecniche per i dettagli.

Estensioni dell'insieme di istruzioni dell'MC68020 e dell'MC68030

Bcc	Gestisce spostamenti di 32 bit
BFxxxx	Istruzioni di campo di bit (BFCHG, BFCLR, BFEXTS, BFEXTU, BFFF0, BFINS, BFSET, BFTST)
BKPT	Funzionalità di nuova istruzione
BRA	Gestisce spostamenti di 32 bit
BSR	Gestisce spostamenti di 32 bit
CALLM	Nuova istruzione (soltanto MC68020)
CAS, CAS2	Nuove istruzioni
CHK	Gestisce operandi di 32 bit
CHK2	Nuova istruzione
CMPI	Gestisce i modi d'indirizzamento relativi al contatore di programma
CMP2	Nuova istruzione
cp	Istruzioni di coprocessore
DIVS/DIVU	Gestisce operandi di 32 bit e di 64 bit
EXTB	Gestisce 8 bit estesi a 32 bit
LINK	Gestisce uno spostamento di 32 bit
MOVEC	Gestisce nuovi registri di controllo
MULS/MULU	Gestisce operandi di 32 bit
PACK	Nuova istruzione
PFLUSH	Istruzione di MMU (soltanto MC68030)
PLOAD	Istruzione di MMU (soltanto MC68030)
PMOVE	Istruzione di MMU (soltanto MC68030)
PTEST	Istruzione di MMU (soltanto MC68030)
RTM	Nuova istruzione (soltanto MC68020)
TST	Gestisce i modi d'indirizzamento relativi al contatore di programma
TRAPcc	Nuova istruzione
UNPK	Nuova istruzione

APPENDICE C

SOMMARIO DEL LINGUAGGIO ASSEMBLER

Quest'appendice presenta le istruzioni del linguaggio assembler dell'assemblatore per l'MC68020 della Motorola. La lista contiene i formati del linguaggio assembler per la CPU, l'unità di gestione della memoria MC68851, e l'unità in virgola mobile MC68881. L'assemblatore della Motorola ed altri prodotti da società diverse per la famiglia di processori dell'MC68020 riconoscono queste istruzioni e le convertono in linguaggio-macchina per il particolare chip interessato.

L'INSIEME DI CARATTERI

L'insieme di caratteri riconosciuto dall'assemblatore strutturato residente per la famiglia M68000 della Motorola è un sottoinsieme del codice ASCII (*American Standard Code for Information Interchange:* codice americano standard per lo scambio di informazioni) del 1968. I caratteri elencati di seguito sono riconosciuti dell'assemblatore. (Il codice ASCII è elencato nell'app. A.)

1. Le lettere maiuscole A-Z
2. Le lettere minuscole a-z (non usate nell'assemblatore per MC68000/010 del VERSAdos)
3. Gli interi 0-9
4. Quattro operatori aritmetici: + – * /
5. Gli operatori logici: >> >> & !
6. Le parentesi tonde ( ), quadre [ ] e graffe { }, impiegate nelle espressioni.

7. I caratteri usati come prefissi speciali:

   # (segno di numero): specifica il modo d'indirizzamento immediato
   $ (segno di dollaro): specifica un numero esadecimale
   @ ("A" commerciale): specifica un numero ottale
   % (segno di percentuale): specifica un numero binario
   ' (apostrofo): specifica un carattere letterale ASCII

8. I caratteri speciali impiegati nelle macro: < > \ @

9. Quattro caratteri di separazione:

   (spazio)
   (tabulazione) (non usata nell'assemblatore per MC68000/010 del VERSAdos)
   , (virgola)
   . (punto)

10. Un commento in un'istruzione-sorgente può includere qualsiasi carattere con valori esadecimali ASCII da 20 (SP) a 7E (~).

11. Un carattere usato come suffisso speciale, il delimitatore delle istruzioni di campo di bit, un delimitatore di CAS2, un delimitatore di divisione troncata, e la fine di un'etichetta:

    : (due punti)

12. Il carattere usato come delimitatore di registro di controllo multiplo:

    | (sbarretta verticale)

L'INSIEME DI ISTRUZIONI

Quest'appendice fornisce un sommario dell'insieme di istruzioni dei processori MC68000/MC68010/MC68020/MC68851/MC68881. Per informazioni dettagliate, si può consultare l'*M68000 16/32-bit Microprocessor Programmer's Reference Manual.* Per l'MC68881 soltanto, i codici di condizione interessati N, Z, I e NAN sono, rispettivamente, i bit 31, 30, 29 e 28 del registro di stato in virgola mobile, anziché i bit 4, 3, 2, 1 e 0 del registro di stato di MC68000/MC68010/MC68020. Quindi i quattro codici di condizione elencati per le istruzioni dell'MC68881 fanno riferimento a N, Z, I e NAN, rispettivamente.

Di seguito sono riportate tre tabelle riepilogative: la prima per l'MC68000/MC68010/MC68020, la seconda per l'MC68851, e la terza per l'MC68881.

SOMMARIO DELL'INSIEME DI ISTRUZIONI - MC68000/MC68010/MC68020

MNEMONICO	DIMENSIONE STANDARD	OPERAZIONE	SINTASSI DI ASSEMBLATORE	CODICI DI CONDIZIONE X N Z V C
ABCD	B	Somma decimale con estensione	ABCD Dy,Dx	* U * U *
			ABCD –(Ay),–(Ax)	
ADD	W	Somma binaria (NOTA 1)	ADD <EA>,Dn	* * * * *
			ADD Dn,<EA>	
ADDA	W	Somma indirizzo	ADDA <EA>,An	- - - - -
ADDI	W	Somma immediata	ADDI #<dato>,<EA>	* * * * *
ADDQ	W	Somma rapida	ADDQ #<dato>,<EA>	* * * * *
ADDX	W	Somma estesa	ADDX Dy,Dx	* * * * *
AND	W	AND logico	AND <EA>,Dn	
			AND Dn,<EA>	- * * 0 0
ANDI	W	AND immediato	ANDI #<dato>,Dy	
ASL,	W	Scorrimento aritmetico	ASd Dx,Dy	* * * * *
ASR			ASd #<dato>,Dy	
			ASd <EA>	
Bcc	W	Salto condizionato	Bcc <etichetta>	- - - - -
BCHG	W	Test di un bit e modifica	BCHG Dn,<EA>	- - * - -
			BCHG #<dato>,<EA>	
BCLR	W	Test di un bit e azzeramento	BCLR Dn,<EA>	- - * - -
			BCLR #<dato>,<EA>	
BFCHG	U	Complemento del campo di bit (MC68020)	BFCHG <EA>(<offset>;<largh>)	- * * 0 0
BFCLR	U	Cancellazione del campo di bit (MC68020)	BFCLR <EA>(<offset>;<largh>)	- * * 0 0
BFEXTS	U	Estrazione del campo di bit con segno (MC68020)	BFEXTS <EA>(<offset>;<largh>),Dn	- * * 0 0
BFEXTU	U	Estrazione del campo di bit senza segno (MC68020)	BFEXTU <EA>(<offset>;<largh>),Dn	- * * 0 0
BFFFO	U	Ricerca del primo in un campo di bit (MC68020)	BFFFO <EA>(<offset>;<largh>),Dn	- * * 0 0
BFINS	U	Inserimento del campo di bit (MC68020)	BFINS <EA>(<offset>;<largh>),Dn	- * * 0 0
BFSET	U	Impostazione del campo di bit (MC68020)	BFESET <EA>(<offset>;<largh>),Dn	- * * 0 0
BFTST	U	Test del campo di bit (MC68020)	BFETST <EA>(<offset>;<largh>),Dn	- * * 0 0

SOMMARIO DELL'INSIEME DI ISTRUZIONI - MC68000/MC68010/MC68020 *(continuazione)*

MNEMONICO	DIMENSIONE STANDARD	OPERAZIONE	SINTASSI DI ASSEMBLATORE	CODICI DI CONDIZIONE X N Z V C
BKPT	U	Breakpoint (MC68020)	BKPT #<dato>	- - - - -
BRA	W	Salto incondizionato	BRA <etichetta>	- - - - -
BSET	L	Test di un bit e attivazione	BSET Dn,<EA> BSET #<dato>,<EA>	- - * - -
BSR	W	Salto a subroutine	BSR <etichetta>	- - - - -
BTST	L	Test di un bit	BTST Dn,<EA> BTST #<dato>,<EA>	- - * - -
CALLM	U	Chiamata di modulo (MC68020)	CALLM #<dato>,<EA>	- - - - -
CAS	W	Confronto e scambio con operando (MC68020)	CAS Dw,Do,<EA>	- * * * *
CAS2	W	Confronto e scambio con operando (MC68020)	CAS2 dw1:dw2,Do1:D02,(Rz1):(Rz2)	- * * * *
CHK	W	Verifica confini di registro	CHK <EA>,Dn	- * U U U
CHK2	W	Verifica confini di registro (MC68020)	CHK2 <EA>,Rn	- U * U *
CLR	W	Azzeramento di un operando	CLR <EA>	- 0 1 0 1
CMP	W	Confronto aritmetico	CMP <EA>,Dn	- * * * *
CMPA	W	Confronto aritmetico di indirizzo	CMPA <EA>,An	- * * * *
CMPI	W	Confronto immediato	CMPI #<dato>,<EA>	- * * * *
CMPM	W	Confronto di memoria	CMPM (Ay)+,(Ax)+	- * * * *
CMP2	W	Confronto di registro entro i limiti (MC68020)	CMP2 <EA>,Rn	- U * U *
DBcc	W	Esame della condizione, decremento e salto (NOTA 2)	DBcc Dn,<etichetta>	- - - - -
DIVS	W	Divisione con segno	DIVS <EA>,Dn	- * * * 0
DIVSL	L	Divisione con segno troncata	DIVSL <EA>,Dr:Dq	- * * * 0
DIVU	W	Divisione senza segno	DIVU <EA>,Dn	- * * * 0
DIVUL	L	Divisione senza segno troncata	DIVUL <EA>,Dr:Dq	- * * * 0

SOMMARIO DELL'INSIEME DI ISTRUZIONI - MC68000/MC68010/MC68020 *(continuazione)*

MNEMONICO	DIMENSIONE STANDARD	OPERAZIONE	SINTASSI DI ASSEMBLATORE	CODICI DI CONDIZIONE X N Z V C
EOR	W	OR esclusivo logico	EOR Dn,<EA>	- * * 0 0
EORI	B/W	OR esclusivo immediato	EORI #<dato>,<EA>	- * * 0 0
EXG	L	Scambio di registri	EXG Rx,Ry	- - - - -
EXT	W	Estensone di segno	EXT Dn	- * * 0 0
EXTB	W	Estensione di segno di byte (MC68020)	EXTB Dn	- * * 0 0
EXTW	W	Estensione di segno di word (MC68020)	EXTW Dn	- * * 0 0
JMP	U*	Salto	JMP <EA>	- - - - -
JSR	U*	Salto a subroutine	JSR <EA>	- - - - -
LEA	L	Caricamento indir. effettivo	LEA <EA>,An	- - - - -
LINK	W	Collegamento e allocazione (NOTA 5)	LINK An,#<spost>	- - - - -
LSL, LSR	W	Scorrimento logico	LSd Dx,Dy LSd #<dato>,Dy LSd <EA>	* * * 0 *
MOVE	W	Trasferimento del dato da sorgente a destinazione	MOVE <EA>,<EA>	- * * 0 0
MOVE a SR	W	Trasf. al registro di stato	MOVE <EA>,SR	* * * * *
MOVE da SR	W	Trasf. dal registro di stato	MOVE SR,<EA>	- - - - -
MOVE a CC	L	Trasf. a codici di condizione	MOVE <EA>,CCR	* * * * *
MOVE da CC	L	Trasf. da codici di condizione (MC68010 o CPU più recenti)	MOVE CCR,<EA>	* * * * *
MOVE USP	L	Trasf. puntatore stack utente	MOVE USP,An MOVE An,USP	- - - - -
MOVEA	W	Trasferimento di indirizzo	MOVEA <EA>,An	- - - - -
MOVEC	L	Trasf. a/da reg. di controllo (MC68010 o più recenti) (NOTA 3)	MOVEC Rc,Rn MOVEC Rn,Rc	- - - - -
MOVEM	W	Trasf. multiplo di registri	MOVEM <lista registri>,<EA> MOVEM <EA>,<lista registri>	- - - - -
MOVEP	W	Trasf. di dati di periferica	MOVEP Dx,d(Ay) MOVEP d(Ay),Dx	- - - - -
MOVEQ	L	Trasferimento rapido	MOVEQ #<dato>,Dn	- * * 0 0
MOVES	W	Trasf. a/da indirizzo (MC68010 o più recenti)	MOVES Rn,<EA>	- - - - -

SOMMARIO DELL'INSIEME DI ISTRUZIONI - MC68000/MC68010/MC68020 *(continuazione)*

MNEMONICO	DIMENSIONE STANDARD	OPERAZIONE	SINTASSI DI ASSEMBLATORE	CODICI DI CONDIZIONE X N Z V C
MULS	W	Moltiplicazione con segno	MULS <EA>,Dn	- * * 0 0
MULU	W	Moltiplicazione senza segno	MULU <EA>,Dn	- * * 0 0
NBCD	B	Negazione decimale con estens.	NBCD <EA>	* U * U *
NEG	W	Negazione in complemento a 2	NEG <EA>	* * * * *
NEGX	W	Negazione con estensione	NEGX <EA>	* * * * *
NOP	U	Nessuna operazione	NOP	- - - - -
NOT	W	Complemento logico	NOT <EA>	- * * 0 0
OR	W	OR logico inclusivo	OR <EA>,Dn	- * * 0 0
			OR Dn,<EA>	
ORI	B/W	OR immediato	ORI #<dato>,<EA>	- * * 0 0
PACK	U	Impaccamento BCD (MC68020)	PACK –(Ax),–(Ay),#<aggiust>	- - - - -
			PACK Dx,Dy,#<aggiust>	
PEA	L	Inserimento dell'indirizzo effettivo nello stack	PEA <EA>	- - - - -
RESET	U	Reset dei dispositivi esterni	RESET	- - - - -
ROL,	W	Rotazione senza estensione	ROd Dx,Dy	- * * 0 *
ROR			ROd #<dato>,Dy	
			ROd, <EA>	
ROXL,	W	Rotazione con estensione	ROXd Dx,Dy	* * * 0 *
ROXR			ROXd #<dato>,Dy	
			ROXd <EA>	
RTD	U	Ritorno da subroutine con spostamento (MC68010 o più recenti) (NOTA 5)	RTD #<spost>	- - - - -
RTE	U	Ritorno da eccezione	RTE	* * * * *
RTM	U	Ritorno da modulo (MC68020)	RTM Rn	- - - - -
RTR	U	Ritorno e ripristino dei codici di condizione	RTR	* * * * *
RTS	U	Ritorno da subroutine	RTS	- - - - -

SOMMARIO DELL'INSIEME DI ISTRUZIONI - MC68000/MC68010/MC68020 *(continuazione)*

MNEMONICO	DIMENSIONE STANDARD	OPERAZIONE	SINTASSI DI ASSEMBLATORE	CODICI DI CONDIZIONE X N Z V C
SBCD	B	Sottraz. decimale con estens.	SBCD Dy,Dx SBCD –(Ay),–(Ax)	* U * U *
Scc	B	Impostazione su condizione	Scc <EA>	- - - - -
STOP	U	Sospensione dell'esecuzione del programma	STOP #<dato>	- - - - -
SUB	W	Sottrazione binaria	SUB <EA>,Dn SUB Dn,<EA>	* * * * *
SUBA	W	Sottrazione di indirizzo	SUBA <EA>,An	- - - - -
SUBI	W	Sottrazione immediata	SUBI #<dato>,<EA>	* * * * *
SUBQ	W	Sottrazione rapida	SUBQ #<dato>,<EA>	* * * * *
SUBX	W	Sottrazione con estensione	SUBX Dy,Dx SUBX –(Ay),–(Ax)	* * * * *
SWAP	W	Scambio di metà del registro	SWAP Dn	- * * 0 0
TAS	B	Test e assegnazione di valore dell'operando	TAS <EA>	- * * 0 0
TRAPcc	U	Trappola sul codice di condizione (MC68020)	TRAPcc TRAPcc.W,#<dato> TRAPcc.L,#<dato>	- - - - -
Tcc	U	Trappola sul codice di condizione (MC68020)	Tcc	- - - - -
TDIVS	W/L	Divisione troncata con segno (MC68020)	TDIVS <EA>,{Di: }Dj	- * * * 0
TDIVU	W/L	Divisione troncata senza segno (MC68020)	TDIVU <EA>,{Di: }Dj	- * * * 0
TMULS	L	Moltiplicazione troncata con segno (MC68020)	TMULS	- * * * 0
TMULU	L	Moltiplicazione troncata senza segno (MC68020)	TMULU	- * * * 0
TPcc	W	Trappola sul codice di condizione (MC68020)	TPcc #xxx	- - - - -
TRAP	U	Trappola	TRAP #<vettore>	- - - - -
TRAPV	U	Trappola su overflow	TRAPV	- - - - -
TST	W	Test di un operando	TST <EA>	- * * 0 0
UNLK	U	Scollegamento	UNLK An	- - - - -
UNPK	U	Disimpaccamento di BCD (MC68020)	UNPK –(Ax),–(Ay),#<aggiust> UNPK Dx,Dy,#<aggiust>	- - - - - - - - - -

SOMMARIO DELL'INSIEME DI ISTRUZIONI - MC68000/MC68010/MC68020 *(continuazione)*

NOTE:

1. <EA> specifica l'indirizzo effettivo (*Effective Address*)
2. L'assemblatore accetta DBRA per la condizione F ("falso").
3. Rc denota un registro di controllo.
4. <lista registri> specifica i registri selezionati per il trasferimento da/verso la memoria.
   <lista registri> può essere:
   Rn: un singolo registro
   Rn – Rm: un insieme di registri consecutivi, con m > n.
   Qualsiasi combinazione delle due possibilità precedenti, separate da una sbarretta (/).
5. <spost> è un intero in complemento a 2, lungo 16 bit, che viene esteso di segno a 32 bit prima di essere aggiunto al puntatore di stack.

6. Le istruzioni BFxxx e TDIVx utilizzano le parentesi graffe { } come richiesto dalla sintassi. In tutti gli altri casi, le parentesi graffe { }, come le quadre [ ], rappresentano termini opzionali.

7. I due punti (:) sono necessari per alcune istruzioni dell'MC68020.
8. Codici dimensionali standard (di *default*):
   B - Byte (.B)
   W - Word (.W)
   L - Longword (.L)
   U - Senza segno (non è ammesso alcun codice di dimensione)
   U* - Normalmente senza segno
9. La sintassi dell'assemblatore per ciascuna istruzione presuppone un'etichetta facoltativa ed un eventuale codice <dimensione>.

SOMMARIO DELL'INSIEME DI ISTRUZIONI - MC68851

MNEMONICO	DIMENSIONE STANDARD	OPERAZIONE	SINTASSI DI ASSEMBLATORE	CODICI DI CONDIZIONE B L S A W I G C
PBcc	W	Salto su condizione di PMMU	PBcc <etichetta>	- - - - - - - -
PDBcc	W	Test, decremento e salto	PDBcc Dn,<etichetta>	- - - - - - - -
PFLUSH	U	Invalidamento entrate in ATC	PFLUSHA (NOTA 1) PFLUSH <fc>,#<maschera> PFLUSH <fc>,#<maschera>,<EA> PFLUSHS <fc>,#<maschera> PFLUSHS <fc>,#<maschera>,<EA>	- - - - - - - -
PFLUSHR	U	Invalidamento di entrate di ATC e RPT	PFLUSHR <EA>	- - - - - - - -
PLOAD	U	Caricamento di entrate nell'ATC	PLOADR <fc>,<EA> (NOTA 1) PLOADW <fc>,<EA>	- - - - - - - -
PMOVE	W	Trasfer. di registro di PMMU	PMOVE Rn,<EA> (NOTA 2) PMOVE <EA>,Rn	* * * * * * * *
PRESTORE	U	Funzione di ripristino PMMU	PRESTORE <EA>	* * * * * * * *
PSAVE	U	Funzione di salvataggio PMMU	PSAVE <EA>	- - - - - - - -
PScc	B	Impostazione su condizione di PMMU	PScc <EA>	- - - - - - - -
PTESTR	U	Acquisizione di informazioni sull'indirizzo logico	PTESTR <fc>,<EA>,#<livello>[,An] PTESTW <fc>,<EA>,#<livello>[,An] (NOTA 1)	* * * * * * * *
PTRAPcc	U/W	Trappola su condizione di PMMU	PTRAPcc PTRAPcc #<dato>	- - - - - - - -
PVALID	L	Convalida di un puntatore	PVALID VAL,<EA> PVALID An,<EA>	- - - - - - - -

NOTE:

1. <fc> è un codice di funzione (*Function Code*) che può assumere uno dei seguenti valori:
   - #<dato> dato immediato
   - Dn registro di dati
   - SFC registro di codice di funzione di sorgente (*Source Function Code*)
   - DFC registro di codice di funzione di destinazione (*Destination Function Code*)

SOMMARIO DELL'INSIEME DI ISTRUZIONI - MC68851 *(continuazione)*

2. I registri Rn della PMMU impiegati in PMOVE sono:

Codice mnemonico	Nome	Dimensione richiesta del codice
TC	Registro di controllo di conversione (*Translation Control register*	L
DRP	Registro del puntatore di radice di DMA (*DMA Root Pointer register*	D
SRP	Registro del puntatore di radice di supervisore (*Supervisor Root Pointer register*	D
CRP	Registro del puntatore di radice di CPU (*CPU Root Pointer register*	D
CAL	Registro del livello di accesso corrente (*Current Access Level register*	B
VAL	Registro del livello di accesso di convalida (*Validate Access Level register*	B
SCC	Registro di controllo del cambio di stack (*Stack Change Control register*	W
BAC0-BAC7	Registri di controllo di riconoscimento di breakpoint (*Breakpoint Acknowledge Control registers*	W
BAD0-BAD7	Registri di dati di riconoscimento di breakpoint (*Breakpoint Acknowledge Data registers*	W
AC	Registro di controllo di accesso (*Access Control register*	W
PSR	Registro di stato della PMMU (*PMMU Status Register*	W
PCSR	Registro di stato del cache della PMMU (*PMMU Cache Status Register*	W

SOMMARIO DELL'INSIEME DI ISTRUZIONI - MC68881

MNEMONICO	DIMENSIONE STANDARD	OPERAZIONE	SINTASSI DI ASSEMBLATORE	CODICI DI CONDIZIONE X N Z V C
FABS	W/X	Funzione di valore assoluto	FABS <EA>,FPn FABS FPm,FPn FABS FPn	* * * * *
FACOS	W/X	Funzione arcocoseno	FACOS <EA>,FPn FACOS FPm,FPn FACOS FPn	* * * * *
FADD	X	Somma in virgola mobile	FADD <EA>,FPn FADD FPm,FPn	* * * * *
FASIN	W/X	Funzione arcoseno	FASIN <EA>,FPn FASIN FPm,FPn FASIN FPn	* * * * *
FATAN	W/X	Funzione arcotangente	FATAN <EA>,FPn FATAN FPm,FPn FATAN FPn	* * * * *
FATANH	W/X	Funzione arcotangente iperbolica	FATANH <EA>,FPn FATANH FPm,FPn FATANH FPn	* * * * *
FBcc	W	Salto condizionato di coprocessore (MC68881)	FBcc <etichetta>	- - - - -
FCMP	X	Confronto in virgola mobile	FCMP <EA>,FPn FCMP FPm,FPn	* * * * *
FCOS	W/X	Funzione coseno	FCOS <EA>,FPn FCOS FPm,FPn FCOS FPn	* * * * *
FCOSH	W/X	Funzione coseno iperbolico	FCOSH <EA>,FPn FCOSH FPm,FPn FCOSH FPn	* * * * *
FDBcc	W	Decremento e salto su condizione (MC68881)	FDBcc Dn,<etichetta>	- - - - -
FDIV	X	Divisione in virgola mobile	FDIV <EA>,FPn FDIV FPm,FPn	* * * * *
FETOX	W/X	Funzione e^x	FETOX <EA>,FPn FETOX FPm,FPn FETOX FPn	* * * * *

SOMMARIO DELL'INSIEME DI ISTRUZIONI - MC68881 *(continuazione)*

MNEMONICO	DIMENSIONE STANDARD	OPERAZIONE	SINTASSI DI ASSEMBLATORE	CODICI DI CONDIZIONE X N Z V C
FETOXM1	W/X	Funzione e^x - 1	FETOXM1 <EA>,FPn FETOXM1 FPm,FPn FETOXM1 FPn	* * * * *
FGETEXP	W/X	Funzione esponenziale	FGETEXP <EA>,FPn FGETEXP FPm,FPn FGETEXP FPn	* * * * *
FGETMAN	W/X	Funzione di mantissa	FGETMAN <EA>,FPn FGETMAN FPm,FPn FGETMAN FPn	* * * * *
FINT	W/X	Funzione di parte intera	FINT <EA>,FPn FINT FPm,FPn FINT FPn	* * * * *
FINTRZ	W/X	Funzione di parte intera arrotondata a zero	FINTRZ <EA>,FPn FINTRZ FPm,FPn FINTRZ FPn	* * * * *
FLOG10	W/X	Funzione di logaritmo in baseo 10	FLOG10 <EA>,FPn FLOG10 FPm,FPn FLOG10 FPn	* * * * *
FLOG2	W/X	Funzione di logaritmo in base 2	FLOG2 <EA>,FPn FLOG2 FPm,FPn FLOG2 FPn	* * * * *
FLOGN	W/X	Funzione di logaritmo naturale	FLOGN <EA>,FPn FLOGN FPm,FPn FLOGN FPn	* * * * *
FLOGNP1	W/X	Funzione log (x + 1)	FLOGNP1 <EA>,FPn FLOGNP1 FPm,FPn FLOGNP1 FPn	* * * * *
FMOD	X	Modulo in virgola mobile	FMOD <EA>,FPn FMOD FPm, FPn	* * * * *
FMOVE	W	Trasferimento a registro in virgola mobile dalla memoria o da un altro registro in virgola mobile	FMOVE <EA>,FPn FMOVE FPm,FPn	* * * * *
	W	Trasferimento da registro in virgola mobile alla memoria	FMOVE FPn,<EA> FMOVE.P FPn,<EA>(#dato) FMOVE.P FPn,<EA>(Dn)	
	L	Trasferimento a registro in virgola mobile dalla memoria o da un altro registro in virgola mobile	FMOVE <EA>,CONTROL|STATUS|IADDR FMOVE CONTROL|STATUS|IADDR,<EA>	

SOMMARIO DELL'INSIEME DI ISTRUZIONI - MC68881 *(continuazione)*

MNEMONICO	DIMENSIONE STANDARD	OPERAZIONE	SINTASSI DI ASSEMBLATORE	CODICI DI CONDIZIONE X N Z V C
FMOVECR	X	Trasferimento di dato di ROM a registro in virgola mobile	FMOVECR #dato,FPn	* * * * *
FMOVEM	X	Trasferimento dalla memoria a più registri in virg. mob.	FMOVEM <EA>,<lista reg. v.m.>	* * * * *
	X	Trasf. a un registro di dati	FMOVEM <EA>,Dn	
			FMOVEM <EA>,CONTROL/STATUS/IADDR	
	L	Trasf. a un registro speciale	FMOVEM <EA>,FPCR/FPSR/FPIAR	
	W	Trasferimento alla memoria da più registri in virg. mob.	FMOVEM <lista reg. v.m.>,<EA>	
	X	Trasf. da registro di dati alla memoria	FMOVEM Dn,<EA> FMOVEM CONTROL/STATUS/IADDR,<EA>	
	L	Trasf. da registro speciale alla memoria	FMOVEM FPCR/FPSR/FPIAR,<EA>	
FMUL	X	Moltiplicazione in virg. mob.	FMUL <EA>,FPn	* * * * *
			FMUL FPm,FPn	
FNEG	W/X	Funzione di negazione	FNEG <EA>,FPn	* * * * *
			FNEG FPm,FPn	
			FNEG FPn	
FNOP	U	NO-OP in virgola mobile	FNOP	* * * * *
FREM	X	Resto in virgola mobile	FREM <EA>,FPn	* * * * *
			FREM FPm,FPn	
FRESTORE	U	Ripristino di stato interno del coprocessore (MC68881) (P)	FRESTORE <EA>	- - - - -
FSAVE	U	Salvataggio di stato interno del coprocessore (MC68881) (P)	FSAVE <EA>	- - - - -
FSCALE	X	Scalamento di esponente in virgola mobile (MC6881) (P)	FSCALE <EA>,FPn FSCALE FPm,FPn	* * * * *
FScc	B	Impostazione su condizione (MC68881)	FScc <EA>	- - - - -
FSGLDIV	X	Divisione di singola precis. in virgola mobile	FSGLDIV <EA>,FPn FSGLDIV FPm,FPn	* * * * * * * * * *
FSGLMUL	X	Moltiplicazione di singola precisione in virg. mob.	FSGLMUL <EA>,FPn FSGLMUL FPm,FPn	* * * * *
FSIN	W/X	Funzione seno	FSIN <EA>,FPn	* * * * *
			FSIN FPm,FPn	
			FSIN FPn	

SOMMARIO DELL'INSIEME DI ISTRUZIONI - MC68881 *(continuazione)*

MNEMONICO	DIMENSIONE STANDARD	OPERAZIONE	SINTASSI DI ASSEMBLATORE	CODICI DI CONDIZIONE X N Z V C
FSINCOS	W/X	Funzione seno/coseno	FISNCOS <EA>,FPm:FPn	* * * * *
FSINH	W/X	Funzione seno iperbolico	FSINH <EA>,FPn FSINH FPm,FPn FSINH FPn	* * * * *
FSQRT	W/X	Funzione radice quadrata	FSQRT <EA>,FPn FSQRT FPm,FPn FSQRT FPn	* * * * *
FSUB	X	Sottrazione in virgola mobile	FSUB <EA>,FPn FSUB FPm,FPn	* * * * *
FTAN	W/X	Funzione tangente	FTAN <EA>,FPn FTAN FPm,FPn FTAN FPn	* * * * *
FTANH	W/X	Funzione tangente iperbolica	FTANH <EA>,FPn FTANH FPm,FPn FTANH FPn	* * * * *
FTENTOX	W/X	Funzione 10^x	FTENTOX <EA>,FPn FTENTOX FPm,FPn FTENTOX FPn	* * * * *
FTcc FTRAPcc	U	Trappola su condizione senza parametro (MC68881)	FTcc FTRAPcc	- - - - -
FTPcc FTRAPcc	W	Trappola su condizione con un parametro (MC68881)	FTPcc #<dato> FTRAPcc #<dato>	- - - - -
FTST	W/X	Test su un operando in virgola mobile	FTST <EA> FTST FPn	* * * * *
FTWOTOX	W/X	Funzione 2^x	FTWOTOX <EA>,FPn FTWOTOX FPm,FPn FTWOTOX FPn	* * * * *
FYTOX	X	Funzione y^x in virg. mob.	FYTOX <EA>,FPn FYTOX FPm,FPn	* * * * *

SOMMARIO DELL'INSIEME DI ISTRUZIONI - MC68881 *(continuazione)*

NOTE:

1. La sintassi dell'assemblatore per ciascuna istruzione presuppone un campo [<etichetta>] facoltativo ed un eventuale campo <dimensione>.

2. Le istruzioni per cui la DIMENSIONE STANDARD è W/X assumono soltanto il valore X quando l'operando è un singolo registro in virgola mobile FPn.

3. Quando il codice "cc" è usato con un'istruzione dell'MC68881, denota un codice di condizione in virgola mobile.

4. (P) denota un'istruzione privilegiata.

APPENDICE D

CARATTERISTICHE DEL LINGUAGGIO-MACCHINA DELL'MC68020

Quest'appendice è tratta dala seconda edizione dell'*MC68020 32-bit Microprocessors User's Manual*, per gentile concessione di Motorola, Inc.

Sono presentate qui le caratteristiche del linguaggio-macchina dell'MC68020. Dapprima sono descritte le espressioni per determinare i codici di condizione per ciascuna istruzione che modifica questi bit; dopodiché, segue la descrizione di ciascuna istruzione, tratta dal manuale citato della Motorola. Di ogni istruzione sono definite l'operazione, il formato e le modalità d'indirizzamento ammesse.

CONDITION CODES COMPUTATION

A.1 INTRODUCTION

This appendix provides a discussion of how the condition codes were developed, the meanings of each bit, how they are computed, and how they are represented in the instruction set details.

Two criteria were used in developing the condition codes:

- Consistency — across instruction, uses, and instances
- Meaningful Results — no change unless it provides useful information

The consistency across instructions means that instructions which are special cases of more general instructions affect the condition codes in the same way. Consistency across instances means that if an instruction ever affects a condition code, it will always affect that condition code. Consistency across uses means that whether the condition codes were set by a compare, test, or move instruction, the conditional instructions test the same situation. The tests used for the conditional instructions and the code computations are given in paragraph A.5.

A.2 CONDITION CODE REGISTER

The condition code register portion of the status register contains five bits:

N — Negative
Z — Zero
V — Overflow
C — Carry
X — Extend

The first four bits are true condition code bits in that they reflect the condition of the result of a processor operation. The X bit is an operand for multiprecision computations. The carry bit (C) and the multiprecision operand extend bit (X) are separate in the M68000 Family to simplify the programming model.

A.3 CONDITION CODE REGISTER NOTATION

In the instruction set details given in **APPENDIX B**, the description of the effect on the condition codes is given in the following form:

Condition Codes:	X	N	Z	V	C

where:

N (negative)	Set if the most significant bit of the result is set. Cleared otherwise.
Z (zero)	Set if the result equals zero. Cleared otherwise.
V (overflow)	Set if there was an arithmetic overflow. This implies that the result is not representable in the operand size. Cleared otherwise.
C (carry)	Set if a carry is generated out of the most significant bit of the operands for an addition. Also, set if a borrow is generated in a subtraction. Cleared otherwise.
X (extend)	Transparent to data movement. When affected, by arithmetic operations, it is set the same as the C bit.

The convention for the notation that is used in the condition code register representation is:

*	set according to the result of the operation
-	not affected by the operation
0	cleared
1	set
U	undefined after the operation

A.4 CONDITION CODE COMPUTATION

Most operations take a source operand and a destination operand, compute, and store the result in the destination location. Unary operations take a destination operand, compute, and store the result in the destination location. Table A-1 details how each instruction sets the condition codes.

A.5 CONDITION TESTS

Table A-2 lists the condition names, encodings, and tests for the condition branch and set instructions. The test associated with each condition is a logical formula based on the current state of the condition codes. If this formula evaluates to one, the condition succeeds, or is true. If the formula evaluates to zero, the condition is unsuccessful, or false. For example, the T condition always succeeds, while the EQ condition succeeds only if the Z bit is currently set in the condition codes.

Table A-1. Condition Code Computations

Operations	X	N	Z	V	C	Special Definition
ABCD	•	U	?	U	?	C = Decimal Carry $Z = Z \wedge \overline{Rm} \wedge \ldots \wedge \overline{R0}$
ADD, ADDI, ADDQ	•	•	•	?	?	$V = Sm \wedge Dm \wedge \overline{Rm} \vee \overline{Sm} \wedge \overline{Dm} \wedge Rm$ $C = Sm \wedge Dm \vee \overline{Rm} \wedge Dm \vee Sm \wedge \overline{Rm}$
ADDX	•	•	?	?	?	$V = Sm \wedge Dm \wedge \overline{Rm} \vee \overline{Sm} \wedge \overline{Dm} \wedge Rm$ $C = Sm \wedge Dm \vee \overline{Rm} \wedge Dm \vee Sm \wedge \overline{Rm}$ $Z = Z \wedge \overline{Rm} \wedge \ldots \wedge \overline{R0}$
AND, ANDI, EOR, EORI, MOVEQ, MOVE, OR, ORI, CLR, EXT, NOT, TAS, TST	—	•	•	0	0	
CHK	—	•	U	U	U	
CHK2, CMP2	—	U	?	U	?	$Z = (R = LB) \vee (R = UB)$ $C = (LB <= UB) \wedge ((R < LB) \vee (R > UB)) \vee$ $(UB < LB) \wedge (R > UB) \wedge (R < LB)$
SUB, SUBI, SUBQ	•	•	•	?	?	$V = \overline{Sm} \wedge Dm \wedge \overline{Rm} \vee Sm \wedge \overline{Dm} \wedge Rm$ $C = Sm \wedge \overline{Dm} \vee Rm \wedge \overline{Dm} \vee Sm \wedge Rm$
SUBX	•	•	?	?	?	$V = \overline{Sm} \wedge Dm \wedge \overline{Rm} \vee Sm \wedge \overline{Dm} \wedge Rm$ $C = Sm \wedge \overline{Dm} \vee Rm \wedge \overline{Dm} \vee Sm \wedge Rm$ $Z = Z \wedge \overline{Rm} \wedge \ldots \wedge \overline{R0}$
CAS, CAS2, CMP, CMPI, CMPM	—	•	•	?	?	$V = \overline{Sm} \wedge Dm \wedge \overline{Rm} \vee Sm \wedge \overline{Dm} \wedge Rm$ $C = Sm \wedge \overline{Dm} \vee Rm \wedge \overline{Dm} \vee Sm \wedge Rm$
DIVS, DIVU	—	•	•	?	0	V = Division Overflow
MULS, MULU	—	•	•	?	0	V = Multiplication Overflow
SBCD, NBCD	•	U	?	U	?	C = Decimal Borrow $Z = Z \wedge \overline{Rm} \wedge \ldots \wedge \overline{R0}$
NEG	•	•	•	?	?	$V = Dm \wedge Rm,\ C = Dm \vee Rm$
NEGX	•	•	?	?	?	$V = Dm \wedge Rm,\ C = Dm \vee Rm$ $Z = Z \wedge \overline{Rm} \wedge \ldots \wedge \overline{R0}$
BTST, BCHG, BSET, BCLR	—	—	?	—	—	$Z = \overline{Dn}$
BFTST, BFCHG, BFSET, BFCLR	—	?	?	0	0	$N = Dm$ $Z = \overline{Dm} \wedge \overline{Dm-1} \wedge \ldots \wedge \overline{D0}$
BFEXTS, BFEXTU, BFFFO	—	?	?	0	0	$N = Sm$ $Z = \overline{Sm} \wedge \overline{Sm-1} \wedge \ldots \wedge \overline{S0}$
BFINS	—	?	?	0	0	$N = Dm$ $Z = \overline{Dm} \wedge \overline{Dm-1} \wedge \ldots \wedge \overline{D0}$
ASL	•	•	•	?	?	$V = Dm \wedge (\overline{Dm-1} \vee \ldots \vee \overline{Dm-r}) \vee \overline{Dm} \wedge (Dm-1 \vee \ldots + Dm-r)$ $C = \overline{Dm-r+1}$
ASL (r = 0)	—	•	•	0	0	
LSL, ROXL	•	•	•	0	?	$C = Dm-r+1$
LSR (r = 0)	—	•	•	0	0	
ROXL (r = 0)	—	•	•	0	?	$C = X$
ROL	—	•	•	0	?	$C = Dm-r+1$
ROL (r = 0)	—	•	•	0	0	
ASR, LSR, ROXR	•	•	•	0	?	$C = Dr-1$
ASR, LSR (r = 0)	—	•	•	0	0	
ROXR (r = 0)	—	•	•	0	?	$C = X$
ROR	—	•	•	0	?	$C = Dr-1$
ROR (r = 0)	—	•	•	0	0	

— = Not Affected
U = Undefined, result meaningless
? = Other – See Special Definition
• = General Case
$X = C$
$N = Rm$
$Z = \overline{Rm} \wedge \ldots \wedge \overline{R0}$
Sm = Source Operand – most significant bit
Dm = Destination Operand – most significant bit
Rm = Result Operand – most significant bit
R = Register Tested
n = Bit Number
r = Shift Count
LB = Lower Bound
UB = Upper Bound
$\wedge$ = Boolean AND
$\vee$ = Boolean OR
$\overline{Rm}$ = NOT Rm

Table A-2. Conditional Tests

Mnemonic	Condition	Encoding	Test
T*	True	0000	1
F*	False	0001	0
HI	High	0010	$\overline{C} \cdot \overline{Z}$
LS	Low or Same	0011	$C + Z$
CC(HS)	Carry Clear	0100	$\overline{C}$
CS(LO)	Carry Set	0101	C
NE	Not Equal	0110	$\overline{Z}$
EQ	Equal	0111	Z
VC	Overflow Clear	1000	$\overline{V}$
VS	Overflow Set	1001	V
PL	Plus	1010	$\overline{N}$
MI	Minus	1011	N
GE	Greater or Equal	1100	$N \cdot V + \overline{N} \cdot \overline{V}$
LT	Less Than	1101	$N \cdot \overline{V} + \overline{N} \cdot V$
GT	Greater Than	1110	$N \cdot V \cdot \overline{Z} + \overline{N} \cdot \overline{V} \cdot \overline{Z}$
LE	Less or Equal	1111	$Z + N \cdot \overline{V} + \overline{N} \cdot V$

• = Boolean AND
+ = Boolean OR
$\overline{N}$ = Boolean NOT N

*Not available for the Bcc instruction

INSTRUCTION SET DETAILS

B.1 INTRODUCTION

This appendix contains detailed information about each instruction in the MC68020 instruction set. They are arranged in alphabetical order with the mnemonic heading set in large bold type for easy reference.

B.2 ADDRESSING CATEGORIES

Effective address modes may be categorized by the ways in which they may be used. The following classifications will be used in the instruction definitions.

- **Data** — If an effective address mode may be used to refer to data operands, it is considered a data addressing effective address mode.
- **Memory** — If an effective address mode may be used to refer to memory operands, it is considered a memory addressing effective address mode.
- **Alterable** — If an effective address mode may be used to refer to alterable (writeable) operands, it is considered an alterable addressing effective address mode.
- **Control** — If an effective address mode may be used to refer to memory operands without an associated size, it is considered a control addressing effective address mode.

Table B-1 shows the various categories to which each of the effective address modes belong.

Table B-1. Effective Addressing Mode Categories

Address Modes	Mode	Register	Data	Memory	Control	Alterable	Assembler Syntax
Data Register Direct	000	reg. no.	X	–	–	X	Dn
Address Register Direct	001	reg. no.	–	–	–	X	An
Address Register Indirect	010	reg. no.	X	X	X	X	(An)
Address Register Indirect with Postincrement	011	reg. no.	X	X	–	X	(An) +
Address Register Indirect with Predecrement	100	reg. no.	X	X	–	X	– (An)
Address Register Indirect with Displacement	101	reg. no	X	X	X	X	(d_{16},An)
Address Register Indirect with Index (8-Bit Displacement)	110	reg. no.	X	X	X	X	(d_8,An,Xn)
Address Register Indirect with Index (Base Displacement)	110	reg. no.	X	X	X	X	(bd,An,Xn)
Memory Indirect Post-Indexed	110	reg. no.	X	X	X	X	([bd,An],Xn,od)
Memory Indirect Pre-Indexed	110	reg. no.	X	X	X	X	([bd,An,Xn],od)
Absolute Short	111	000	X	X	X	X	(xxx).W
Absolute Long	111	001	X	X	X	X	(xxx).L
Program Counter Indirect with Displacement	111	010	X	X	X	–	(d_{16},PC)
Program Counter Indirect with Index (8-Bit Displacement)	111	011	X	X	X	–	(d_8,PC,Xn)
Program Counter Indirect with Index (Base Displacement)	111	011	X	X	X	–	(bd,PC,Xn)
PC Memory Indirect Post-Indexed	111	011	X	X	X	–	([bd,PC],Xn,od)
PC Memory Indirect Pre-Indexed	111	011	X	X	X	–	([bd,PC,Xn],od)
Immediate	111	100	X	X	–	–	#<data>

These categories may be combined so that additional, more restrictive, classifications may be defined. For example, the instruction descriptions use such classifications as alterable memory or data alterable. The former refers to those addressing modes which are both alterable and memory addresses, and the latter refers to addressing modes which are both data and alterable.

B.3 INSTRUCTION DESCRIPTION

The formats of each instruction are given in the following ~~pages~~. Figure B-1 illustrates what information is given.

Instruction Name ——→ **ABCD** Add

Operation Description (see paragraph B.4) ——→ **Operation:** $Source_{10}$ + Destinatio

Assembler Syntax for this Instruction ——→ **Assembler Syntax:** ABCD Dy,Dx / ABCD −(Ay), −(Ax)

Attributes: Size = (Byte)

Text Description of Instruction Operation ——→ **Description:** Add the source operar bit, and store the result in the d binary coded decimal arithmeti ways:
1. Data register to data registe specified in the instruction.
2. Memory to memory: The oper ing mode using the address
This operation is a byte operati

Condition Code Effects (see **Appendix A**) ——→ **Condition Codes:**

X	N	Z	V	C
*	U	*	U	*

N Undefined.
Z Cleared if the result is n
V Undefined.
C Set if a carry (decimal) w
X Set the same as the carr

Normally the Z conditior an operation. This allows multiple-precision opera

Instruction Format — Specifies the bit pattern and fields of the operation word and any other words which are part of the instruction. The effective address extensions are not explicitly illustrated. The extensions (if there are any) would follow the illustrated portions of the instructions. For the MOVE instruction, the source effective address extension is the first, followed by the destination effective address extension. ——→ **Instruction Format:**

15	14	13	12	11 10 9
1	1	0	0	Register Rx

Meanings and allowed values of the various fields required by the instruction format. ——→ **Instruction Fields:**
Register Rx field — Specifies t
If R/M = 0, specifies a data re
If R/M = 1, specifies an addre
R/M field — Specifies the oper
0 — The operation is data re
1 — The operation is memor

Figure B-1. Instruction Description Format

B.4 OPERATION DESCRIPTION DEFINITIONS

The following definitions are used for the operation description in the details of the instruction set.

OPERANDS:

An	— address register
Dn	— data register
Rn	— any data or address register
PC	— program counter
SR	— status register
CCR	— condition codes (lower order byte of status register)
SSP	— supervisor stack pointer
USP	— user stack pointer
SP	— active stack pointer (equivalent to A7)
X	— extend operand (from condition codes)
Z	— zero condition code
V	— overflow condition code
Immediate Data	— immediate data from the instruction
d	— address displacement
Source	— source contents
Destination	— destination contents
Vector	— location of exception vector
ea	— any valid effective address

SUBFIELDS AND QUALIFIERS:

<bit>OF<operand>	selects a single bit of the operand
<ea>{offset:width}	selects a bit field
(<operand>)	the contents of the referenced location
<operand>10	the operand is binary coded decimal; operations are to be performed in decimal.
(<address register>) −(<address register>) (<address register>)+	the register indirect operator which indicates that the operand register points to the memory location of the instruction operand. The optional mode qualifiers are −, +, (d) and (d, ix); these are explained in **SECTION 2 DATA ORGANIZATION AND ADDRESSING CAPABILITIES.**
#xxx or #<data>	immediate data located with the instruction is the operand.

OPERATIONS: Operations are grouped into binary, unary, and other.

Binary—These operations are written <operand><op><operand> where <op> is one of the following:

→	the left operand is moved to the right operand
↔	the two operands are exchanged
+	the operands are added
−	the right operand is subtracted from the left operand
*	the operands are multiplied
/	the first operand is divided by the second operand
Λ	the operands are logically ANDed
V	the operands are logically ORed
⊕	the operands are logically exclusively ORed
<	relational test, true if left operand is less than right operand
>	relational test, true if left operand is greater than right operand
shifted by	the left operand is shifted or rotated by the number of positions
rotated by	specified by the right operand

Unary:

~ <operand>	the operand is logically complemented
<operand>sign-extended	the operand is sign extended, all bits of the upper portion are made equal to high order bit of the lower portion
<operand>tested	the operand is compared to 0, the results are used to set the condition codes

Other:

TRAP equivalent to Format/Offset Word→(SSP); SSP − 2→SSP; PC→(SSP); SSP − 4→SSP; SR→(SSP); SSP − 2→SSP; (vector)→PC

STOP enter the stopped state, waiting for interrupts

If <condition> then <operations> else <operations>. The condition is tested. If true, the operations after the "then" are performed. If the condition is false and the optional "else" clause is present, the operations after the "else" are performed. If the condition is false and the optional "else" clause is absent, the instruction performs no operation.

ABCD — Add Decimal with Extend — ABCD

Operation: $Source_{10} + Destination_{10} + X \rightarrow Destination$

Assembler Syntax:
ABCD Dy,Dx
ABCD −(Ay), −(Ax)

Attributes: Size = (Byte)

Description: Add the source operand to the destination operand along with the extend bit, and store the result in the destination location. The addition is performed using binary coded decimal arithmetic. The operands may be addressed in two different ways:

1. Data register to data register: The operands are contained in the data registers specified in the instruction.
2. Memory to memory: The operands are addressed with the predecrement addressing mode using the address registers specified in the instruction.

This operation is a byte operation only.

Condition Codes:

X	N	Z	V	C
*	U	*	U	*

N Undefined.
Z Cleared if the result is non-zero. Unchanged otherwise.
V Undefined.
C Set if a carry (decimal) was generated. Cleared otherwise.
X Set the same as the carry bit.

NOTE

Normally the Z condition code bit is set via programming before the start of an operation. This allows successful tests for zero results upon completion of multiple-precision operations.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	Register Rx			1	0	0	0	0	R/M	Register Ry		

Instruction Fields:

Register Rx field — Specifies the destination register:
If R/M = 0, specifies a data register
If R/M = 1, specifies an address register for the predecrement addressing mode

R/M field — Specifies the operand addressing mode:
0 — The operation is data register to data register
1 — The operation is memory to memory

Register Ry field — Specifies the source register:
If R/M = 0, specifies a data register
If R/M = 1, specifies an address register for the predecrement addressing mode

ADD — Add — ADD

Operation: Source + Destination → Destination

Assembler Syntax: ADD <ea>,Dn
ADD Dn,<ea>

Attributes: Size = (Byte, Word, Long)

Description: Add the source operand to the destination operand using binary addition, and store the result in the destination location. The size of the operation may be specified to be byte, word, or long. The mode of the instruction indicates which operand is the source and which is the destination as well as the operand size.

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

- N Set if the result is negative. Cleared otherwise.
- Z Set if the result is zero. Cleared otherwise.
- V Set if an overflow is generated. Cleared otherwise.
- C Set if a carry is generated. Cleared otherwise.
- X Set the same as the carry bit.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	Register Dn			Op-Mode			Effective Address Mode			Effective Address Register		

Instruction Fields:

Register field — Specifies any of the eight data registers.

Op-Mode field —

Byte	Word	Long	Operation
000	001	010	<ea> + <Dn> → <Dn>
100	101	110	<Dn> + <ea> → <ea>

Effective Address Field — Determines addressing mode:

a. If the location specified in a source operand, the all addressing modes are allowed as shown:

ADD — Add — ADD

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An*	001	reg. number:An
(An)	010	reg. number:An
(An)+	011	reg. number:An
−(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Word and Long only.

b. If the location specified is a destination operand, then only alterable memory addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
−(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

Notes: 1. If the destination is a data register, then it cannot be specified by using the destination <ea> mode, but must use the destination Dn mode instead.
2. ADDA is used when the destination is an address register. ADDI and ADDQ are used when the source is immediate data. Most assemblers automatically make this distinction.

ADDA — Add Address — ADDA

Operation: Source + Destination → Destination

Assembler Syntax: ADDA <ea>,An

Attributes: Size = (Word, Long)

Description: Add the source operand to the destination address register, and store the result in the address register. The size of the operation may be specified to be word or long. The entire destination address register is used regardless of the operation size.

Condition Codes: Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	Register An			Op-Mode			Effective Address Mode			Effective Address Register		

Instruction Fields:

Register field — Specifies any of the eight address registers. This is always the destination.

Op-Mode field — Specifies the size of the operation:
- 011—word operation. The source operand is sign-extended to a long operand and the operation is performed on the address register using all 32 bits.
- 111—long operation.

Effective Address field — Specifies the source operand. All addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	001	reg. number:An
(An)	010	reg. number:An
(An)+	011	reg. number:An
−(An)	100	reg. number:An
$(d_{16}$,An)	101	reg. number:An
$(d_8$,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
$(d_{16}$,PC)	111	010
$(d_8$,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

ADDI — Add Immediate — ADDI

Operation: Immediate Data + Destination → Destination

Assembler Syntax: ADDI #<data>,<ea>

Attributes: Size = (Byte, Word, Long)

Description: Add the immediate data to the destination operand, and store the result in the destination location. The size of the operation may be specified to be byte, word, or long. The size of the immediate data matches the operation size.

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

- N Set if the result is negative. Cleared otherwise.
- Z Set if the result is zero. Cleared otherwise.
- V Set if an overflow is generated. Cleared otherwise.
- C Set if a carry is generated. Cleared otherwise.
- X Set the same as the carry bit.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	1	0	Size		Effective Address Mode			Effective Address Register		
Word Data								Byte Data							
Long Data (Includes Previous Word)															

Instruction Fields:

Size field — Specifies the size of the operation:
- 00—byte operation.
- 01—word operation.
- 10—long operation.

Effective Address field — Specifies the destination operand. Only data alterable addressing modes are allowed as shown:

ADDI — Add Immediate — ADDI

Addr. Mode	Mode	Register	Addr. Mode	Mode	Register
Dn	000	reg. number:Dn	(xxx).W	111	000
An	—	—	(xxx).L	111	001
(An)	010	reg. number:An	#<data>	—	—
(An)+	011	reg. number:An			
−(An)	100	reg. number:An			
(d_{16},An)	101	reg. number:An	(d_{16},PC)	—	—
(d_8,An,Xn)	110	reg. number:An	(d_8,PC,Xn)	—	—
(bd,An,Xn)	110	reg. number:An	(bd,PC,Xn)	—	—
([bd,An,Xn],od)	110	reg. number:An	([bd,PC,Xn],od)	—	—
([bd,An],Xn,od)	110	reg. number:An	([bd,PC],Xn,od)	—	—

Immediate field — (Data immediately following the instruction):
If size = 00, then the data is the low order byte of the immediate word.
If size = 01, then the data is the entire immediate word.
If size = 10, then the data is the next two immediate words.

ADDQ — Add Quick — ADDQ

Operation: Immediate Data + Destination → Destination

Assembler Syntax: ADDQ #<data>,<ea>

Attributes: Size = (Byte, Word, Long)

Description: Add the immediate data to the operand at the destination location. The data range is from 1 to 8. The size of the operation may be specified to be byte, word, or long. Word and long operations are also allowed on the address registers, in which case the condition codes are not affected. When adding to address registers, the entire destination address register is used, regardless of the operation size.

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

N Set if the result is negative. Cleared otherwise.
Z Set if the result is zero. Cleared otherwise.
V Set if an overflow is generated. Cleared otherwise.
C Set if a carry is generated. Cleared otherwise.
X Set the same as the carry bit.

The condition codes are not affected if the destination is an address register.

Instruction Format:

15	14	13	12	11 10 9	8	7 6	5 4 3	2 1 0
							Effective Address	
0	1	0	1	Data	0	Size	Mode	Register

Instruction Fields:
Data field — Three bits of immediate data, 0, 1-7 representing a range of 8, 1 to 7 respectively.
Size field — Specifies the size of the operation:
00—byte operation.
01—word operation.
10—long operation.
Effective Address field — Specifies the destination location. Only alterable addressing modes are allowed as shown:

ADDQ — Add Quick — ADDQ

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An*	001	reg. number:An
(An)	010	reg. number:An
(An)+	011	reg. number:An
−(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

*Word and Long Only.

ADDX — Add Extended — ADDX

Operation: Source + Destination + X → Destination

Assembler Syntax:
ADDX Dy,Dx
ADDX −(Ay), −(Ax)

Attributes: Size = (Byte, Word, Long)

Description: Add the source operand to the destination operand along with the extend bit and store the result in the destination location. The operands may be addressed in two different ways:
1. Data register to data register: the operands are contained in data registers specified in the instruction.
2. Memory to memory: the operands are addressed with the predecrement addressing mode using the address registers specified in the instruction.

The size of the operation may be specified to be byte, word, or long.

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

N Set if the result is negative. Cleared otherwise.
Z Cleared if the result is non-zero. Unchanged otherwise.
V Set if an overflow is generated. Cleared otherwise.
C Set if a carry is generated. Cleared otherwise.
X Set the same as the carry bit.

NOTE

Normally the Z condition code bit is set via programming before the start of an operation. This allows successful tests for zero results upon completion of multiple-precision operations.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	Register Rx			1	Size		0	0	R/M	Register Ry		

ADDX — Add Extended — ADDX

Instruction Fields:

Register Rx field — Specifies the destination register:
- If R/M = 0, specifies a data register.
- If R/M = 1, specifies an address register for the predecrement addressing mode.

Size field — Specifies the size of the operation:
- 00—byte operation.
- 01—word operation.
- 10—long operation.

R/M field — Specifies the operand address mode:
- 0—The operation is data register to data register.
- 1—The operation is memory to memory.

Register Ry field — Specifies the source register:
- If R/M = 0, specifies a data register.
- If R/M = 1, specifies an address register for the predecrement addressing mode.

AND — AND Logical — AND

Operation: Source∧Destination → Destination

Assembler Syntax: AND <ea>,Dn
AND Dn,<ea>

Attributes: Size = (Byte, Word, Long)

Description: AND the source operand to the destination operand and store the result in the destination location. The size of the operation may be specified to be byte, word, or long. The contents of an address register may not be used as an operand.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

- N Set if the most significant bit of the result is set. Cleared otherwise.
- Z Set if the result is zero. Cleared otherwise.
- V Always cleared.
- C Always cleared.
- X Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	Register Dn			Op-Mode			Effective Address Mode			Effective Address Register		

Instruction Fields:

Register field — Specifies any of the eight data registers.

Op-Mode field —

Byte	Word	Long	Operation
000	001	010	(<ea>)∧(<Dn>) → Dn
100	101	110	(<Dn>)∧(<ea>) → ea

Effective Address field — Determines addressing mode:
If the location specified is a source operand then only data addressing modes are allowed as shown:

AND — AND Logical — AND

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
−(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

If the location specified is a destination operand then only alterable memory addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
−(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

Notes: 1. If the destination is a data register, then it cannot be specified by using the destination <ea> mode, but must use the destination Dn mode instead.
2. ANDI is used when the source is immediate data. Most assemblers automatically make this distinction.

ANDI — AND Immediate — ANDI

Operation: Immediate Data∧Destination → Destination

Assembler Syntax: ANDI #<data>,<ea>

Attributes: Size = (Byte, Word, Long)

Description: AND the immediate data to the destination operand and store the result in the destination location. The size of the operation may be specified to be byte, word, or long. The size of the immediate data matches the operation size.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

N Set if the most significant bit of the result is set. Cleared otherwise.
Z Set if the result is zero. Cleared otherwise.
V Always cleared.
C Always cleared.
X Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	Size		Effective Address Mode			Register		
Word Data								Byte Data							
Long Data (Includes Previous Word)															

Instruction Fields:
Size field — Specifies the size of the operation:
00—byte operation.
01—word operation.
10—long operation.
Effective Address field — Specifies the destination operand. Only data alterable addressing modes are allowed as shown:

ANDI — AND Immediate — ANDI

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
−(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

Immediate field — (Data immediately following the instruction):
If size = 00, then the data is the low order byte of the immediate word.
If size = 01, then the data is the entire immediate word.
If size = 10, then the data is the next two immediate words.

ANDI to CCR — AND Immediate to Condition Codes — ANDI to CCR

Operation: Source ∧ CCR → CCR

Assembler Syntax: ANDI #<data>,CCR

Attributes: Size = (Byte)

Description: AND the immediate operand with the condition codes and store the result in the low-order byte of the status register.

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

N Cleared if bit 3 of immediate operand is zero. Unchanged otherwise.
Z Cleared if bit 2 of immediate operand is zero. Unchanged otherwise.
V Cleared if bit 1 of immediate operand is zero. Unchanged otherwise.
C Cleared if bit 0 of immediate operand is zero. Unchanged otherwise.
X Cleared if bit 4 of immediate operand is zero. Unchanged otherwise.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	0	0	1	1	1	1	0	0
0	0	0	0	0	0	0	0	Byte Data (8 Bits)							

ANDI to SR — AND Immediate to the Status Register (Privileged Instruction) — ANDI to SR

Operation: If supervisor state
then Source ∧ SR → SR
else TRAP

Assembler Syntax: ANDI #<data>,SR

Attributes: Size = (Word)

Description: AND the immediate operand with the contents of the status register and store the result in the status register. All bits of the status register are affected.

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

N Cleared if bit 3 of immediate operand is zero. Unchanged otherwise.
Z Cleared if bit 2 of immediate operand is zero. Unchanged otherwise.
V Cleared if bit 1 of immediate operand is zero. Unchanged otherwise.
C Cleared if bit 0 of immediate operand is zero. Unchanged otherwise.
X Cleared if bit 4 of immediate operand is zero. Unchanged otherwise.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	0	1	1	1	1	1	0	0
Word Data (16 Bits)															

ASL, ASR — Arithmetic Shift — ASL, ASR

Operation: Destination Shifted by <count> → Destination

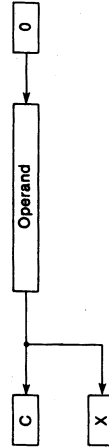
Assembler Syntax: ASd Dx,Dy
ASd #<data>,Dy
ASd <ea>
where d is direction, L or R

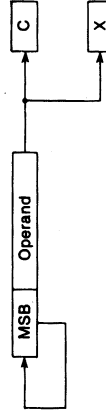
Attributes: Size = (Byte, Word, Long)

Description: Arithmetically shift the bits of the operand in the direction (L or R) specified. The carry bit receives the last bit shifted out of the operand. The shift count for the shifting of a register may be specified in two different ways:
1. Immediate: the shift count is specified in the instruction (shift range, 1-8).
2. Register: the shift count is contained in a data register specified in the instruction (shift count is modulo 64).

The size of the operation may be specified to be byte, word, or long. The content of memory may be shifted one bit only, and the operand size is restricted to a word.

For ASL, the operand is shifted left; the number of positions shifted is the shift count. Bits shifted out of the high order bit go to both the carry and the extend bits; zeroes are shifted into the low order bit. The overflow bit indicates if any sign changes occur during the shift.

ASL: C ← Operand ← 0; X ←

For ASR, the operand is shifted right; the number of positions shifted is the shift count. Bits shifted out of the low order bit go to both the carry and the extend bits; the sign bit (MSB) is replicated into the high order bit.

ASR: MSB Operand → C; → X

ASL, ASR — Arithmetic Shift — ASL, ASR

Condition Codes:

X	N	Z	V	X
*	*	*	*	*

N Set if the most significant bit of the result is set. Cleared otherwise.
Z Set if the result is zero. Cleared otherwise.
V Set if the most significant bit is changed at any time during the shift operation. Cleared otherwise.
C Set according to the last bit shifted out of the operand. Cleared for a shift count of zero.
X Set according to the last bit shifted out of the operand. Unaffected for a shift count of zero.

Instruction Format (Register Shifts):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	Count Register			dr	Size		i/r	0	0	Register		

Instruction Fields (Register Shifts):

Count/Register field — Specifies shift count or register where count is located:
If i/r = 0, the shift count is specified in this field. The values 0, 1-7 represent a range of 8, 1 to 7 respectively.
If i/r = 1, the shift count (modulo 64) is contained in the data register specified in this field.

dr field — Specifies the direction of the shift:
0—shift right.
1—shift left.

Size field — Specifies the size of the operation:
00—byte operation.
01—word operation.
10—long operation.

i/r field —
If i/r = 0, specifies immediate shift count.
If i/r = 1, specifies register shift count.

Register field — Specifies a data register whose content is to be shifted.

ASL, ASR — Arithmetic Shift — ASL, ASR

Instruction Format (Memory Shifts):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	0	0	dr	1	1	Effective Address Mode			Register		

Instruction Fields (Memory Shifts):

dr field — Specifies the direction of the shift:
0—shift right
1—shift left

Effective Address field — Specifies the operand to be shifted. Only memory alterable addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
−(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

Bcc — Branch Conditionally — Bcc

Operation: If (condition true) then PC + d → PC

Assembler Syntax: Bcc <label>

Attributes: Size = (Byte, Word, Long)

Description: If the specified condition is met, program execution continues at location (PC) + displacement. The displacement is a twos complement integer which counts the relative distance in bytes. The value in the PC is the sign-extended instruction location plus two. If the 8-bit displacement in the instruction word is zero, then the 16-bit displacement (word immediately following the instruction) is used. If the 8-bit displacement in the instruction word is all ones ($FF), then the 32-bit displacement (long word immediately following the instruction) is used. "cc" may specify the following conditions:

CC	carry clear	0100	$\overline{C}$
CS	carry set	0101	C
EQ	equal	0111	Z
GE	greater or equal	1100	$N \cdot V + \overline{N} \cdot \overline{V}$
GT	greater than	1110	$N \cdot V \cdot \overline{Z} + \overline{N} \cdot \overline{V} \cdot \overline{Z}$
HI	high	0010	$\overline{C} \cdot \overline{Z}$
LE	less or equal	1111	$Z + N \cdot \overline{V} + \overline{N} \cdot V$

LS	low or same	0011	$C + Z$
LT	less than	1101	$N \cdot \overline{V} + \overline{N} \cdot V$
MI	minus	1011	N
NE	not equal	0110	$\overline{Z}$
PL	plus	1010	$\overline{N}$
VC	overflow clear	1000	$\overline{V}$
VS	overflow set	1001	V

Condition Codes: Not affected.

Instruction Format:

15	14	13	12	11–8	7–0
0	1	1	0	Condition	8-Bit Displacement
16-Bit Displacement if 8-Bit Displacement = $00					
32-Bit Displacement if 8-Bit Displacement = $FF					

Instruction Fields:

Condition field — One of fourteen conditions discussed in description.

8-Bit Displacement field — Twos complement integer specifying the relative distance (in bytes) between the branch instruction and the next instruction to be executed if the condition is met.

16-Bit Displacement field — Allows a larger displacement than 8 bits. Used only if the 8-bit displacement is equal to $00.

32-Bit Displacement field — Allows a larger displacement than 16 bits. Used only if the 8-bit displacement is equal to $FF.

Note: A short branch to the immediately following instruction cannot be generated, because it would result in a zero offset, which forces a word branch instruction definition.

BCHG — Test a Bit and Change — BCHG

Operation: ~(<bit number> of Destination) → Z;
~(<bit number> of Destination) → <bit number> of Destination

Assembler Syntax: BCHG Dn,<ea>
BCHG #<data>,<ea>

Attributes: Size = (Byte, Long)

Description: A bit in the destination operand is tested and the state of the specified bit is reflected in the Z condition code. After the test, the state of the specified bit is changed in the destination. If a data register is the destination, then the bit numbering is modulo 32 allowing bit manipulation on all bits in a data register. If a memory location is the destination, a byte is read from that location, the bit operation is performed using the bit number, modulo 8, and the byte is written back to the location. In all cases, bit zero refers to the least significant bit. The bit number for this operation may be specified in two different ways:

1. Immediate — the bit number is specified in a second word of the instruction.
2. Register — the bit number is contained in a data register specified in the instruction.

Condition Codes:

X	N	Z	V	C
—	—	*	—	—

N Not affected.
Z Set if the bit tested is zero. Cleared otherwise.
V Not affected.
C Not affected.
X Not affected.

Instruction Format (Bit Number Dynamic specified by a register):

15	14	13	12	11–9	8	7	6	5–3	2–0
								Effective Address	
0	0	0	0	Register Dn	1	0	1	Mode	Register

Instruction Fields (Bit Number Dynamic):

Register field — Specifies the data register whose content is the bit number.

Effective Address field — Specifies the destination location. Only data alterable addressing modes are allowed as shown:

BCHG Test a Bit and Change BCHG

Addr. Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
-(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

*Long only; all others are byte only.

Instruction Format (Bit Number Static, specified as immediate data):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	0	1	Effective Address Mode			Register		
0	0	0	0	0	0	0	0	Bit Number							

Instruction Fields (Bit Number Static):

Effective Address field — Specifies the destination location. Only data alterable addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
-(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

*Long only; all others are byte only.

Bit Number field — Specifies the bit number.

BCLR Test a Bit and Clear BCLR

Operation: ~(<bit number> of Destination) → Z;
0 → <bit number> of Destination

Assembler Syntax: BCLR Dn,<ea>
BCLR #<data>,<ea>

Attributes: Size = (Byte, Long)

Description: A bit in the destination operand is tested and the state of the specified bit is reflected in the Z condition code. After the test, the specified bit is cleared in the destination. If a data register is the destination, then the bit numbering is modulo 32 allowing bit manipulation on all bits in a data register. If a memory location is the destination, a byte is read from that location, the bit operation performed using the bit number, modulo 8, and the byte written back to the location. In all cases, bit zero refers to the least significant bit. The bit number for this operation may be specified in two different ways:

1. Immediate — the bit number is specified in a second word of the instruction.
2. Register — the bit number is contained in a data register specified in the instruction.

Condition Codes:

X	N	Z	V	C
—	—	*	—	—

N Not affected.
Z Set if the bit tested is zero. Cleared otherwise.
V Not affected.
C Not affected.
X Not affected.

Instruction Format (Bit Number Dynamic, specified in a register):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	Register Dn			1	1	0	Effective Address Mode			Register		

Instruction Fields (Bit Number Dynamic):

Register field — Specifies the data register whose content is the bit number.

Effective Address field — Specifies the destination location. Only data alterable addressing modes are allowed as shown:

BCLR — Test a Bit and Clear — BCLR

Addr. Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
−(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

*Long only; all others are byte only

Instruction Format (Bit Number Static, specified as immediate data):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	1	0	Effective Address Mode			Register		
0	0	0	0	0	0	0	0	Bit Number							

Instruction Fields (Bit Number Static):

Effective Address field — Specifies the destination location. Only data alterable addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
−(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	000	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

*Long only; all others are byte only.

Bit Number field — Specifies the bit number.

BFCHG — Test Bit Field and Change — BFCHG

Operation: ~(<bit field> of Destination) → <bit field> of Destination

Assembler Syntax: BFCHG <ea> {offset:width}

Attributes: Unsized

Description: Complement a bit field at the specified effective address location. The condition codes are set according to the value in the field before it is changed.

The field selection is specified by a field offset and field width. The field offset denotes the starting bit of the field. The field width determines the number of bits to be included in the field.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

- N Set if the most significant bit of the field is set. Cleared otherwise.
- Z Set if all bits of the field are zero. Cleared otherwise.
- V Always cleared.
- C Always cleared.
- X Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	0	1	0	1	1	Effective Address Mode			Register		
0	0	0	0	Do	Offset					Dw	Width				

Instruction Fields:

Effective Address field — Specifies the base location for the bit field. Only data register direct or alterable control addressing modes are allowed, as shown below:

Do field — Determines how the field offset is specified.
- 0—the field offset is in the Offset field.
- 1—bits [8:6] of the extension word specify a data register which contains the offset; bits [10:9] are 0.

Offset field — Specifies the field offset, depending on Do.
- If Do = 0—the Offset field is an immediate operand; the operand value is in the range 0-31, specifying a field offset of 0-31.
- If Do = 1—the Offset field specifies a data register which contains the offset. The value is in the range -2^{31} to $2^{31}-1$.

BFCHG — Test Bit Field and Change — BFCHG

Dw field — Determines how the field width is specified.
0—the field width is in the Width field.
1—bits [2:0] of the extension word specify a data register which contains the width; bits [3:4] are 0.
Width field — Specifies the field width, depending on Dw.
If Dw = 0—the Width field is an immediate operand; the operand value is in the range 0, 1-31 specifying a field width of 32, 1-31 respectively.
If Dw = 1—the Width field specifies a data register which contains the width. The operand value is taken modulo 32, with values 0, 1-31 specifying a field width of 32, 1-31.

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	—	—
−(An)	—	—
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

BFCLR — Test Bit Field and Clear — BFCLR

Operation: 0 → <bit field> of Destination

Assembler Syntax: BFCLR <ea> {offset:width}

Attributes: Unsized

Description: Clear a bit field at the specific effective address location. The condition codes are set according to the value in the field before it is cleared.

The field selection is specified by a field offset and field width. The field offset denotes the starting bit of the field. The field width determines the number of bits to be included in the field.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

N Set if the most significant bit of the field is set. Cleared otherwise.
Z Set if all bits of the field are zero. Cleared otherwise.
V Always cleared.
C Always cleared.
X Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	1	0	0	1	1	Effective Address Mode			Register		
0	0	0	0	Do	Offset					Dw	Width				

Instruction Fields:
Effective Address field — Specifies the base location for the bit field. Only data register direct or alterable control addressing modes are allowed, as shown below:
Do field — Determines how the field offset is specified.
0—the field offset is in the Offset field.
1—bits [8:6] of the extension word specify a data register which contains the offset; bits [10:9] are 0.
Offset field — Specifies the field offset, depending on Do.
If Do = 0—the Offset field is an immediate operand; the operand value is in the range 0-31, specifying a field offset of 0-31.
If Do = 1—the Offset field specifies a data register which contains the offset. The value is in the range -2^{31} to $2^{31}-1$.

BFCLR Test Bit Field and Clear BFCLR

Dw field — Determines how the field width is specified.
0—the field width is in the Width field.
1—bits [2:0] of the extension word specify a data register which contains the width; bits [4:3] are 0.
Width field — Specifies the field width, depending on Dw.
If Dw = 0—the Width field is an immediate operand; the operand value is in the range 0, 1-31 specifying a field width of 32, 1-31.
If Dw = 1—the Width field specifies a data register which contains the width. The operand value is taken modulo 32, with values 0, 1-31 specifying a field width of 32, 1-31.

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	—	—
–(An)	—	—
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
	•	
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

BFEXTS Extract Bit Field Signed BFEXTS

Operation: <bit field> of Source → Dn

Assembler Syntax: BFEXTS <ea> {offset:width},Dn

Attributes: Unsized

Description: Extract a bit field from the specified effective address location, sign extend to 32 bits, and load the result into the destination data register.

The field selection is specified by a field offset and field width. The field offset denotes the starting bit of the field. The field width determines the number of bits to be included in the field.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

N Set if the most significant bit of the field is set. Cleared otherwise.
Z Set if all bits of the field are zero. Cleared otherwise.
V Always cleared.
C Always cleared.
X Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	0	1	1	1	1	Effective Address Mode			Effective Address Register		
0	Register			Do	Offset					Dw	Width				

Instruction Fields:
Effective Address field — Specifies the base location for the bit field. Only data register direct or control addressing modes are allowed as shown below:
Register field — Specifies the destination register.
Do field — Determines how the field offset is specified.
0—the field offset is in the Offset field.
1—bits [8:6] of the extension word specify a data register which contains the offset; bits [10:9] are 0.
Offset field — Specifies the field offset, depending on Do.
If Do = 0—the Offset field is an immediate operand; the operand value is in the range 0-31, specifying a field offset of 0-31.
If Do = 1—the Offset field specifies a data-register which contains the offset. The value is in the range -2^{31} to $2^{31}-1$.

BFEXTS — Extract Bit Field Signed — BFEXTS

Dw field — Determines how the field width is specified.
- 0—the field width is in the Width field.
- 1—bits [2:0] of the extension word specify a data register which contains the width; bits [4:3] are 0.

Width field — Specifies the field width, depending on Dw.
- If Dw = 0—the Width field is an immediate operand; the operand value is in the range 0, 1-31, specifying a field width of 32, 1-31.
- If Dw = 1—the Width field specifies a data register which contains the width. The operand value is taken modulo 32, with values 0, 1-31 specifying a field width of 32, 1-31.

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	—	—
−(An)	—	—
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

BFEXTU — Extract Bit Field Unsigned — BFEXTU

Operation: <bit field> of Source → Dn

Assembler Syntax: BFEXTU <ea> {offset:width},Dn

Attributes: Unsized

Description: Extract a bit field from the specified effective address location, zero extend to 32 bits, and load the results into the destination data register.

The field selection is specified by a field offset and field width. The field offset denotes the starting bit of the field. The field width determines the number of bits to be included in the field.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

- N Set if the most significant bit of the source field is set. Cleared otherwise.
- Z Set if all bits of the field are zero. Cleared otherwise.
- V Always cleared.
- C Always cleared.
- X Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	0	0	1	1	1	Effective Address: Mode			Register		
0	Register			Do	Offset					Dw	Width				

Instruction Fields:

Effective Address field — Specifies the base location for the bit field. Only data register direct or control addressing modes are allowed as shown below:

Register field — Specifies the destination data register.

Do field — Determines how the field offset is specified.
- 0—the field offset is in the Offset field.
- 1—bits [8:6] of the extension word specify a data register which contains the offset; bits [10:9] are 0.

Offset field — Specifies the field offset, depending on Do.
- If Do = 0—the Offset field is an immediate operand; the operand value is in the range 0-31, specifying a field offset of 0-31.
- If Do = 1—the Offset field specifies a data register which contains the offset. The value is in the range -2^{31} to $2^{31}-1$.

BFEXTU Extract Bit Field Unsigned BFEXTU

Dw field — Determines how the field width is specified.
  0—the field width is in the Width field.
  1—bits [2:0] of the extension word specify a data register which contains the width; bits [4:3] are 0.
Width field — Specifies the field width, depending on Dw.
  If Dw = 0—the Width field is an immediate operand; the operand value is in the range 0, 1-31, specifying a field width of 32, 1-31.
  If Dw = 1—the Width field specifies a data register which contains the width. The operand value is taken modulo 32, with values 0, 1-31 specifying a field width of 32, 1-31.

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	—	—
-(An)	—	—
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

BFFFO Find First One in Bit Field BFFFO

Operation: <bit field> of Source Bit Scan → Dn

Assembler Syntax: BFFFO <ea> {offset:width},Dn

Attributes: Unsized

Description: The source operand is searched for the most significant bit position that contains a set bit. The bit offset (the original bit offset plus the offset of the first set bit) of that bit is then placed in Dn. If no bit of the bit field is set, the value placed in Dn is the field offset plus field width. The condition codes are set according to the bit field operand.

The field selection is specified by a field offset and field width. The field offset denotes the starting bit of the field. The field width determines the number of bits to be included in the field.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

N Set if the most significant bit of the field is set. Cleared otherwise.
Z Set if all bits of the field are zero. Cleared otherwise.
V Always cleared.
C Always cleared.
X Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	1	0	1	1	1	Effective Address Mode			Effective Address Register		
0	Register			Do	Offset					Dw	Width				

Instruction Fields:
Effective Address field — Specifies the base location for the bit field. Only data register direct or control addressing modes are allowed as shown below:
Register field — Specifies the destination data register operand.
Do field — Determines how the field offset is specified.
  0—the field offset is in the Offset field.
  1—bits [8:6] of the extension word specify a data register which contains the offset; bits [10:9] are 0.
Offset field — Specifies the field offset, depending on Do.
  If Do = 0—the Offset field is an immediate operand; the operand value is in the range 0-31, specifying a field offset of 0-31.
  If Do = 1—the Offset field specifies a data register which contains the offset. The value is in the range -2^{31} to $2^{31}-1$.

BFFFO — Find First One in Bit Field — BFFFO

Dw field — Determines how the field width is specified.
0—the field width is in the Width field.
1—bits [2:0] of the extension word specify a data register which contains the width; bits [4:3] are 0.

Width field — Specifies the field width, depending on Dw.
If Dw = 0—the Width field is an immediate operand; the operand value is in the range 0, 1-31, specifying a field width of 32, 1-31.
If Dw = 1—the Width field specifies a data register which contains the width. The operand value is taken modulo 32, with values 0, 1-31 specifying a field width of 32, 1-31.

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	—	—
-(An)	—	—
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

BFINS — Insert Bit Field — BFINS

Operation: Dn → <bit field> of Destination

Assembler Syntax: BFINS Dn,<ea> {offset:width}

Attributes: Unsized

Description: Move a bit field from the low-order bits of the specified data register to a bit field at the specified effective address location. The condition codes are set according to the inserted value.

The field selection is specified by a field offset and field width. The field offset denotes the starting bit of the field. The field width determines the number of bits to be included in the field.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

N Set if the most significant bit of the field is set. Cleared otherwise.
Z Set if all bits of the field are zero. Cleared otherwise.
V Always cleared.
C Always cleared.
X Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	1	1	1	1	1	Effective Address Mode			Effective Address Register		
0	Register			Do	Offset					Dw	Width				

Instruction Fields:

Effective Address field — Specifies the base location for the bit field. Only data register direct or alterable control addressing modes are allowed as shown below:

Register field — Specifies the source data register operand.

Do field — Determines how the field offset is specified.
0—the field offset is in the Offset field.
1—bits [8:6] of the extension word specify a data register which contains the offset; bits [10:9] are 0.

Offset field — Specifies the field offset, depending on Do.
If Do = 0—the Offset field is an immediate operand; the operand value is in the range 0-31, specifying a field offset of 0-31.
If Do = 1—the Offset field specifies a data register which contains the offset. The value is in the range -2^{31} to $2^{31}-1$.

BFINS Insert Bit Field BFINS

Dw field — Determines how the field width is specified.
- 0—the field width is in the Width field.
- 1—bits [2:0] of the extension word specify a data register which contains the width; bits [4:3] are 0.

Width field — Specifies the field width, depending on Dw.
- If Dw = 0—the Width field is an immediate operand; the operand value is in the range 0, 1-31, specifying a field width of 32, 1-31.
- If Dw = 1—the Width field specifies a data register which contains the width. The operand value is taken modulo 32, with values 0, 1-31 specifying a field width of 32, 1-31.

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	—	—
−(An)	—	—
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

BFSET Set Bit Field BFSET

Operation: 1s → <bit field> of Destination

Assembler Syntax: BFSET <ea> {offset:width}

Attributes: Unsized

Description: Set all bits of a bit field at the specified effective address location. The condition codes are set according to the value in the field before it is set.

The field selection is specified by a field offset and field width. The field offset denotes the starting bit of the field. The field width determines the number of bits to be included in the field.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

- N Set if the most significant bit of the field is set. Cleared otherwise.
- Z Set if all bits of the field are zero. Cleared otherwise.
- V Always cleared.
- C Always cleared.
- X Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	1	1	0	1	1	Effective Address Mode			Effective Address Register		
0	0	0	0	Do	Offset					Dw	Width				

Instruction Fields:

Effective Address field — Specifies the base location for the bit field. Only data register direct or alterable control addressing modes are allowed as shown below:

Do field — Determines how the field offset is specified.
- 0—the field offset is in the Offset field.
- 1—bits [8:6] of the extension word specify a data register which contains the offset; bits [10:9] are 0.

Offset field — Specifies the field offset, depending on Do.
- If Do = 0—the Offset field is an immediate operand; the operand value is in the range 0-31, specifying a field offset of 0-31.
- If Do = 1—the Offset field specifies a data register which contains the offset. The value is in the range -2^{31} to $2^{31}-1$.

BFSET — Set Bit Field — BFSET

Dw field — Determines how the field width is specified.
  0—the field width is in the Width field.
  1—bits [2:0] of the extension word specify a data register which contains the width; bits [4:3] are 0.
Width field — Specifies the field width, depending on Dw.
  If Dw = 0—the Width field is an immediate operand; the operand value is in the range 0, 1-31, specifying a field width of 32, 1-31.
  If Dw = 1—the Width field specifies a data register which contains the width. The operand value is taken modulo 32, with values 0, 1-31 specifying a field width of 32, 1-31.

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	—	—
−(An)	—	—
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

BFTST — Test Bit Field — BFTST

Operation: <bit field> of Destination

Assembler Syntax: BFTST <ea> {offset:width}

Attributes: Unsized

Description: Extract a bit field from the specified effective address location, and set the condition codes according to the value in the field.

The field selection is specified by a field offset and field width. The field offset denotes the starting bit of the field. The field width determines the number of bits to be included in the field.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

N Set if the most significant bit of the field is set. Cleared otherwise.
Z Set if all bits of the field are zero. Cleared otherwise.
V Always cleared.
C Always cleared.
X Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	0	0	0	1	1	Effective Address Mode			Effective Address Register		
0	0	0	0	Do	Offset					Dw	Width				

Instruction Fields:
Effective Address field — Specifies the base location for the bit field. Only data register direct or control addressing modes are allowed as shown below:
Do field — Determines how the field offset is specified.
  0—the field offset is in the Offset field.
  1—bits [8:6] of the extension word specify a data register which contains the offset; bits [10:9] are 0.
Offset field — Specifies the field offset, depending on Do.
  If Do = 0—the Offset field is an immediate operand; the operand value is in the range 0-31, specifying a field offset of 0-31.
  If Do = 1—the Offset field specifies a data register which contains the offset. The value is in the range -2^{31} to $2^{31}-1$.

BFTST — Test Bit Field — BFTST

Dw field — Determines how the field width is specified.
0—the field width is in the Width field.
1—bits [2:0] of the extension word specify a data register which contains the width; bits [4:3] are 0.

Width field — Specifies the field width, depending on Dw.
If Dw = 0—the Width field is an immediate operand; the operand value is in the range 0, 1-31, specifying a field width of 32, 1-31.
If Dw = 1—the Width field specifies a data register which contains the width. The operand value is taken modulo 32, with values 0, 1-31 specifying a field width of 32, 1-31.

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	—	—
-(An)	—	—
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

BKPT — Breakpoint — BKPT

Operation: If breakpoint vector acknowledged
then execute returned operation word
else Trap as illegal instruction

Assembler Syntax: BKPT #<data>

Attributes: Unsized

Description: This instruction is used to support the program breakpoint function for debug monitors and real-time hardware emulators, and the operation will be dependent on the implementation. Execution of this instruction will cause the MC68020 to run a breakpoint acknowledge bus cycle, with the immediate data (value 0-7) presented on address lines A2, A3, and A4, and zeros on address lines A0 and A1. Two responses are permitted: normal and exception.

The normal response to the MC68020 is an operation word (typically an instruction, originally replaced by the breakpoint instruction) on the data lines with the $\overline{DSACKx}$ signal asserted. This operation word will then be executed in place of the breakpoint instruction.

For the exception response, a bus error signal will cause the MC68020 to take an illegal instruction exception.

Condition Codes: Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	1	0	0	1	Vector		

Instruction Fields:

Vector field — Specifies the breakpoint for which the processor is to request the corresponding operation word.

BRA — Branch Always — BRA

Operation: PC + d → PC

Assembler Syntax: BRA <label>

Attributes: Size = (Byte, Word, Long)

Description: Program execution continues at location (PC) + displacement. The displacement is a twos complement integer, which counts the relative distance in bytes. The value in the PC is the instruction location plus two. If the 8-bit displacement in the instruction word is zero, then the 16-bit displacement (word immediately following the instruction) is used. If the 8-bit displacement in the instruction word is all ones ($FF), then the 32-bit displacement (long word immediately following the instruction) is used.

Condition Codes: Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	0	8-Bit Displacement							
16-Bit Displacement if 8-Bit Displacement = $00															
32-Bit Displacement if 8-Bit Displacement = $FF															

Instruction Fields:

8-Bit Displacement field — Two complement integer specifying the relative distance (in bytes) between the branch instruction and the next instruction to be executed.

16-Bit Displacement field — Allows a larger displacement than 8 bits. Used only if the 8-bit displacement is equal to $00.

32-Bit Displacement field — Allows a larger displacement than 8 bits. Used only if the 8-bit displacement is equal to $FF.

Note: A short branch to the immediately following instruction cannot be generated because it would result in a zero offset, which forces a word branch instruction definition.

BSET — Test a Bit and Set — BSET

Operation: ~(<bit number> of Destination) → Z;
1 → <bit number> of Destination

Assembler Syntax: BSET Dn,<ea>
BSET #<data>,<ea>

Attributes: Size = (Byte, Long)

Description: A bit in the destination operand is tested, and the state of the specified bit is reflected in the Z condition code. After the test, the specified bit is set in the destination. If a data register is the destination, then the bit numbering is modulo 32, allowing bit manipulation on all bits in a data register. If a memory location is the destination, a byte is read from that location, the bit operation performed using the bit number, modulo 8, and the byte written back to the location. Bit zero refers to the least significant bit. The bit number for this operation may be specified in two different ways:

1. Immediate — the bit number is specified in a second word of the instruction.
2. Register — the bit number is contained in a data register specified in the instruction.

Condition Codes:

X	N	Z	V	C
—	—	*	—	—

- N Not affected.
- Z Set if the bit tested is zero. Cleared otherwise.
- V Not affected.
- C Not affected.
- X Not affected.

Instruction Format (Bit Number Dynamic, specified in a register):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
										Effective Address					
0	0	0	0	Register			1	1	1	Mode			Register		

Instruction Fields (Bit Number Dynamic):

Register field — Specifies the data register whose content is the bit number.

Effective Address field — Specifies the destination location. Only data alterable addressing modes are allowed as shown:

BSET — Test a Bit and Set — BSET

Addr. Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
-(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

*Long only; all others are byte only.

Instruction Format (Bit Number Static, specified as immediate data):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	1	1	Effective Address Mode			Effective Address Register		
0	0	0	0	0	0	0	0	Bit Number							

Instruction Fields (Bit Number Static):
Effective Address field — Specifies the destination location. Only data alterable addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
-(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

*Long only; all others are byte only.

Bit Number field — Specifies the bit number.

BSR — Branch to Subroutine — BSR

Operation: SP − 4 → SP; PC → (SP); PC + d → PC

Assembler Syntax: BSR <label>

Attributes: Size = (Byte, Word, Long)

Description: The long word address of the instruction immediately following the BSR instruction is pushed onto the system stack. Program execution then continues at location (PC) + displacement. The displacement in a twos complement integer which counts the relative distances in the bytes. The value in the PC is the instru.:ion location plus two. If the 8-bit displacement in the instruction word is zero, then the 16-bit displacement (word immediately following the instruction) is used. If the 8-bit displacement in the instruction word is all ones ($FF), then the 32-bit displacement (long word immediately following the instruction) is used.

Condition Codes: Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	1	8-Bit Displacement							
16-Bit Displacement if 8-Bit Displacement = $00															
32-Bit Displacement if 8-Bit Displacement = $FF															

Instruction Fields:
8-Bit Displacement field — Twos complement integer specifying the relative distance (in bytes) between the branch instruction and the next instruction to be executed.
16-Bit Displacement field — Allows a larger displacement than 8 bits. Used only if the 8-bit displacement is equal to $00.
32-Bit Displacement field — Allows a larger displacement than 8 bits. Used only if the 8-bit displacement is equal to $FF.

Note: A short subroutine branch to the immediately following instruction cannot be generated because it would result in a zero offset, which forces a word branch instruction definition.

BTST — Test a Bit — BTST

Operation: ~(<bit number> of Destination)→Z;

Assembler Syntax:
BTST Dn,<ea>
BTST #<data>,<ea>

Attributes: Size=(Byte, Long)

Description: A bit in the destination operand is tested, and the state of the specified bit is reflected in the Z condition code. If a data register is the destination, then the bit numbering is modulo 32, allowing bit manipulation on all bits in a data register. If a memory location is the destination, a byte is read from that location, and the bit operation performed using the bit number, modulo 8, with zero referring to the least significant bit. The bit number for this operation may be specified in two different ways:
1. Immediate — the bit number is specified in a second word of the instruction.
2. Register — the bit number is contained in a data register specified in the instruction.

Condition Codes:

X	N	Z	V	C
—	—	*	—	—

- N Not affected.
- Z Set if the bit tested is zero. Cleared otherwise.
- V Not affected.
- C Not affected.
- X Not affected.

Instruction Format (Bit Number Dynamic, specified in a register):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	Register Dn			1	0	0	Effective Address Mode			Register		

Instruction Fields (Bit Number Dynamic):
Register field — Specifies the data register whose content is the bit number.
Effective Address field — Specifies the destination location. Only data addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
−(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Long only; all others are byte only.

Instruction Format (Bit Number Static, specified as immediate data):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	0	0	Effective Address Mode			Register		
0	0	0	0	0	0	0	0	Bit Number							

Instruction Fields (Bit Number Static):
Effective Address field — Specifies the destination location. Only data addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
−(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*Long only; all others are byte only.

Bit Number field — Specifies the bit number.

CALLM — CALL Module — CALLM

Operation: Save current module state on stack;
Load new module state from destination

Assembler Syntax: CALLM #<data>,<ea>

Attributes: Unsized

Description: The effective address of the instruction is the location of an external module descriptor. A module frame is created on the top of the stack, and the current module state is saved in the frame. The immediate operand specifies the number of bytes of arguments to be passed to the called module. A new module state is loaded from the descriptor addressed by the effective address.

Condition Codes: Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	1	0	1	1	Effective Address Mode			Register		
0	0	0	0	0	0	0	0	Argument Count							

Instruction Fields:

Effective Address field — Specifies the address of the module descriptor. Only control addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An)+	—	—
−(An)	—	—
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

Agrument Count field — Specifies the number of bytes of arguments to be passed to the called module. The 8-bit field can specify from 0 to 255 bytes of arguments. The same number of bytes is removed from the stack by the RTM instruction.

CAS / CAS2 — Compare and Swap with Operand — CAS / CAS2

Operation: CAS Destination — Compare Operand → cc;
if Z, Update_Operand → Destination
else Destination → Compare_Operand

CAS2 Destination 1 — Compare 1 → cc;
if Z, Destination 2 — Compare 2 → cc
if Z, Update 1 → Destination 1 Update 2 → Destination 2
else Destination 1 → Compare 1 Destination 2 → Compare 2

Assembler Syntax: CAS Dc,Du,<ea>
CAS2 Dc1:Dc2,Du1:Du2,(Rn1):(Rn2)

Attributes: Size = (Byte*, Word, Long)

Description: The Effective Address operand(s) is fetched and compared to the compare operand data register(s). If the operands match, the update operand data register(s) is (are) written to the destination location(s); otherwise, the memory operand location is left unchanged and the compare operand is loaded with the memory operand. The operation is indivisible (using a read-modify-write memory cycle) to allow synchronization of several processors.

Condition Codes:

X	N	Z	V	C
—	*	*	*	*

N Set if the result is negative. Cleared otherwise.
Z Set if the result is zero. Cleared otherwise.
V Set if an overflow is generated. Cleared otherwise.
C Set if a carry is generated. Cleared otherwise.
X Not affected.

Instruction Format: (Single Operand):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	Size		0	1	1	Effective Address Mode			Register		
0	0	0	0	0	0	0	Du			0	0	0	Dc		

Instruction Fields:

Size field — Specifies the size of the operation.
01—byte operation.
10—word operation.
11—long operation.

Effective Address field — Specifies the location of the tested operand. Only alterable memory addressing modes are allowed as shown below:

Du field — Specifies the data register which holds the update value to be written to the memory operand location if the comparison is successful.

*Single Operand Form Only

CAS / CAS2 — Compare and Swap with Operand — CAS / CAS2

Dc field — Specifies the data register which contains the test value to be compared against the memory operand.

Addr. Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
-(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

Instruction Format (Dual Operand):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	Size		0	1	1	1	1	1	1	•0	0
D/A1	Rn1			0	0	0	Du1			0	0	0	Dc1		
D/A2	Rn2			0	0	0	Du2			0	0	0	Dc2		

Instruction Fields:

Size field — Specifies the size of the operation.
10—word operation.
11—long operation.

D/A1,D/A2 fields — Specify whether Rn1 and Rn2 reference data or address registers, respectively.
0—The corresponding register is a data register.
1—The corresponding register is an address register.

Rn1,Rn2 fields — Specify the numbers of the registers which contain the address of the first and second tested operands, respectively. If the operands overlap in memory, the results of any memory update are undefined.

Du1,Du2 fields — Specify the data registers which hold the update values to be written to the first and second memory operand locations if the comparison is successful.

Dc1,Dc2 fields — Specify the data registers which contain the test values to be compared against the first and second memory operands, respectively. If Dc1 and Dc2 specify the same data register and the comparison fails, the data register is loaded from the first memory operand.

Programming Note: The CAS and CAS2 instructions may be used to perform secure update operations on system control data structures in a multiprocessing environment.

CHK — Check Register Against Bounds — CHK

Operation: If Dn <0 or Dn> Source then TRAP

Assembler Syntax: CHK <ea>,Dn

Attributes: Size = (Word, Long)

Description: The content of the data register specified in the instruction is examined and compared to the upper bound. The upper bound is a twos complement integer. If the register value is less than zero or greater than the upper bound, then the processor initiates exception processing. The vector number is generated to reference the CHK instruction exception vector.

Condition Codes:

X	N	Z	V	C
—	*	U	U	U

N Set if Dn< 0; cleared if Dn> Source. Undefined otherwise.
Z Undefined.
V Undefined.
C Undefined.
X Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	Register Dn			Size			Effective Address Mode			Effective Address Register		

Instruction Fields:

Register field — Specifies the data register whose content is checked.

Size field — Specifies the size of the operation.
110—word operation.
100—long operation.

Effective Address field — Specifies the upper bound operand word. Only data addressing modes are allowed as shown:

CHK — Check Register Against Bounds — CHK

Addr. Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
−(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

CHK2 — Check Register Against Bounds — CHK2

Operation: If Rn<Source—lower-bound or
Rn>Source—upper-bound
Then TRAP

Assembler Syntax: CHK2 <ea>,Rn

Attributes: Size = (Byte, Word, Long)

Description: Check the value in Rn against the bounds pair at the effective address location. The lower bound is at the address specified by the effective address, with the upper bound at that address plus the operand length. For signed comparisons, the arithmetically smaller value should be the lower bound, while for unsigned comparison, the logically smaller value should be the lower bound.

The size of the data to be checked, and the bounds to be used, may be specified as byte, word, or long. If the checked register is a data register and the operation size is byte or word, only the appropriate low-order part of Rn is checked. If the checked register is an address register and the operation size is byte or word, the bounds operands are sign-extended to 32 bits and the resultant operands compared against the full 32 bits of An.

If the upper bound equals the lower bound, then the valid range is a single value. If the register operand is out of bounds, the processor initiates exception processing. The vector number is generated to reference the CHK instruction exception vector. Otherwise, the next instruction is executed.

Condition Codes:

X	N	Z	V	C
—	U	*	U	*

N Undefined.
Z Set if Rn is equal to either bound. Cleared otherwise.
V Undefined.
C Set if Rn is out of bounds. Cleared otherwise.
X Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	Size		0	1	1	Effective Address Mode			Register		
D/A	Register			1	0	0	0	0	0	0	0	0	0	0	0

CHK2 — Check Register Against Bounds — CHK2

Instruction Fields:

Size field — Specifies the size of the operation.
00—byte operation.
01—word operation.
10—long operation.

Effective Address field — Specifies the location of the bounds operands. Only control addressing modes are allowed as shown below:

D/A field — Specifies whether an address register or data register is to be checked.
0—Data register.
1—Address register.

Register field — Specifies the address or data register whose content is to be checked.

Addr. Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An)+	—	—
−(An)	—	—
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

CLR — Clear an Operand — CLR

Operation: 0 → Destination

Assembler Syntax: CLR <ea>

Attributes: Size = (Byte, Word, Long)

Description: The destination is cleared to all zero. The size of the operation may be specified to be byte, word, or long.

Condition Codes:

X	N	Z	V	C
—	0	1	0	0

N Always cleared.
Z Always set.
V Always cleared.
C Always cleared.
X Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	1	0	Size		Effective Address Mode			Effective Address Register		

Instruction Fields:

Size field — Specifies the size of the operation.
00—byte operation.
01—word operation.
10—long operation.

Effective Address field — Specifies the destination location. Only data alterable addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn*	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
−(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

CMP — Compare — CMP

Operation: Destination — Source

Assembler Syntax: CMP <ea>,Dn

Attributes: Size = (Byte, Word, Long)

Description: Subtract the source operand from the specified data register and set the condition codes according to the result; the data register is not changed. The size of the operation may be byte, word, or long.

Condition Codes:

X	N	Z	V	C
—	*	*	*	*

N Set if the result is negative. Cleared otherwise.
Z Set if the result is zero. Cleared otherwise.
V Set if an overflow is generated. Cleared otherwise.
C Set if a borrow is generated. Cleared otherwise.
X Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	Register Dn			Op-Mode			Effective Address Mode			Register		

Instruction Fields:

Register field — Specifies the destination data register.

Op-Mode field —

Byte	Word	Long	Operation
000	001	010	Dn – (<ea>)

Effective Address field — Specifies the source operand. All addressing modes are allowed as shown:

CMP — Compare — CMP

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An*	001	reg. number:An
(An)	010	reg. number:An
(An)+	011	reg. number:An
–(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

*Word and Long only.

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

Note: CMPA is used when the destination is an address register. CMPI is used when the source is immediate data. CMPM is used for memory to memory compares. Most assemblers automatically make this distinction.

CMPA — Compare Address — CMPA

Operation: Destination – Source

Assembler Syntax: CMPA <ea>,An

Attributes: Size = (Word, Long)

Description: Subtract the source operand from the destination address register and set the condition codes according to the result; the address register is not changed. The size of the operation may be specified to be word or long. Word length source operands are sign extended to 32-bit quantities before the operation is done.

Condition Codes:

X	N	Z	V	C
—	*	*	*	*

N Set if the result is negative. Cleared otherwise.
Z Set if the result is zero. Cleared otherwise.
V Set if an overflow is generated. Cleared otherwise.
C Set if a borrow is generated. Cleared otherwise.
X Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	Register An			Op-Mode			Effective Address Mode			Effective Address Register		

Instruction Fields:

Register field — Specifies the destination data register.
Op-Mode field — Specifies the size of the operation:
011—word operation. The source operand is sign-extended to a long operand and the operation is performed on the address register using all 32 bits.
111—long operation.
Effective Address field — Specifies the source operand. All addressing modes are allowed as shown:

CMPA — Compare Address — CMPA

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	001	reg. number:An
(An)	010	reg. number:An
(An)+	011	reg. number:An
–(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

CMPI — Compare Immediate — CMPI

Operation: Destination – Immediate Data

Assembler Syntax: CMPI #<data>,<ea>

Attributes: Size = (Byte, Word, Long)

Description: Subtract the immediate data from the destination operand and set the condition codes according to the result; the destination location is not changed. The size of the operation may be specified to be byte, word, or long. The size of the immediate data matches the operation size.

Condition Codes:

X	N	Z	V	C
—	*	*	*	*

- N Set if the result is negative. Cleared otherwise.
- Z Set if the result is zero. Cleared otherwise.
- V Set if an overflow is generated. Cleared otherwise.
- C Set if a borrow is generated. Cleared otherwise.
- X Not affected.

Instruction Format:

<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">1</td><td rowspan="2">1</td><td rowspan="2">0</td><td rowspan="2">0</td><td colspan="2" rowspan="2">Size</td><td colspan="6">Effective Address</td></tr>
<tr><td colspan="3">Mode</td><td colspan="3">Register</td></tr>
<tr><td colspan="8">Word Data</td><td colspan="8">Byte Data</td></tr>
<tr><td colspan="16">Long Data</td></tr>
</table>

Instruction Fields:

Size field — Specifies the size of the operation:
- 00—byte operation.
- 01—word operation.
- 10—long operation.

Effective Address field — Specifies the destination operand. Only data addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
–(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

Immediate field — (Data immediately following the instruction):
- If size = 00, then the data is the low order byte of the immediate word.
- If size = 01, then the data is the entire immediate word.
- If size = 10, then the data is the next two immediate words.

CMPM — Compare Memory — CMPM

Operation: Destination – Source

Assembler Syntax: CMPM (Ay)+,(Ax)+

Attributes: Size = (Byte, Word, Long)

Description: Subtract the source operand from the destination operand, and set the condition codes according to the results; the destination location is not changed. The operands are always addressed with the postincrement addressing mode, using the address registers specified in the instruction. The size of the operation may be specified to be byte, word, or long.

Condition Codes:

X	N	Z	V	C
—	*	*	*	*

- N Set if the result is negative. Cleared otherwise.
- Z Set if the result is zero. Cleared otherwise.
- V Set if an overflow is generated. Cleared otherwise.
- C Set if a borrow is generated. Cleared otherwise.
- X Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	Register Ax			1	Size		0	0	1	Register Ay		

Instruction Fields:

Register Ax field — (always the destination) Specifies an address register for the postincrement addressing mode.

Size field — Specifies the size of the operation:
- 00—byte operation.
- 01—word operation.
- 10—long operation.

Register Ay field — (always the source) Specifies an address register for the postincrement addressing mode.

CMP2 — Compare Register Against Bounds — CMP2

Operation: Compare Rn<Source—lower-bound or
Rn>Source—upper-bound
and Set Condition Codes

Assembler Syntax: CMP2 <ea>,Rn

Attributes: Size = (Byte, Word, Long)

Description: Compare the value in Rn against the bounds pair at the effective address location and set the condition codes accordingly. The lower bound is at the address specified by the effective address, with the upper bound at that address plus the operand length. For signed comparisons, the arithmetically smaller value should be the lower bound, while for unsigned comparison, the logically smaller value should be the lower bound.

The size of the data to be compared, and the bounds to be used, may be specified as byte, word, or long. If the compared register is a data register and the operation size is byte or word, only the appropriate low-order part of Dn is checked. If the checked register is an address register and the operation size is byte or word, the bounds operands are sign-extended to 32 bits and the resultant operands compared against the full 32 bits of An.

If the upper bound equals the lower bound, then the valid range is a single value.

NOTE: This instruction is analougous to CHK2, but avoids causing exception processing to handle the out-of-bounds case.

Condition Codes:

X	N	Z	V	C
—	U	*	U	*

- N Undefined.
- Z Set if Rn is equal to either bound. Cleared otherwise.
- V Undefined.
- C Set if Rn is out of bounds. Cleared otherwise.
- X Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
										Effective Address					
										Mode			Register		
0	0	0	0	0	Size		0	1	1						
D/A	Register			0	0	0	0	0	0	0	0	0	0	0	0

CMP2 — Compare Register Against Bounds — CMP2

Instruction Fields:

Size field — Specifies the size of the operation.
- 00—byte operation.
- 01—word operation.
- 10—long operation.

Effective Address field — Specifies the location of the bounds pair. Only control addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An)+	—	—
-(An)	—	—
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

D/A field — Specifies whether an address register or data register is to be compared.
- 0—Data register.
- 1—Address register.

Register field — Specifies the address or data register whose content is to be checked.

cpBcc — Branch on Coprocessor Condition — cpBcc

Operation: If cpcc true then PC + d → PC

Assembler Syntax: cpBcc <label>

Attributes: Size = (Word, Long)

Description: If the specified coprocessor condition is met, program execution continues at location (PC) + displacement. The displacement is a twos complement integer which counts the relative distance in bytes. The value in the PC is the address of the displacement word(s). The displacement may be either 16 bits or 32 bits. The coprocessor determines the specific condition from the condition field in the operation word.

Condition Codes: Not affected.

Instruction Format:

15	14	13	12	11–9	8	7	6	5–0
1	1	1	1	Cp-Id	0	1	W/L	Coprocessor Condition
Optional Coprocessor Defined Extension Words								
Word or								
Long Word Displacement								

Instruction Fields:

Cp-Id field — Identifies the coprocessor that is to process this operation.

W/L field — Specifies the size of the displacement.
- 0—the displacement is 16 bits.
- 1—the displacement is 32 bits.

Coprocessor Condition field — Specifies the coprocessor condition to be tested. This field is passed to the coprocessor, which provides directives to the main processor for processing this instruction.

16-Bit Displacement field — The shortest displacement form for coprocessor branches is 16 bits.

32-Bit Displacement field — Allows a displacement larger than 16 bits.

cpDBcc Test Coprocessor Condition Decrement and Branch cpDBcc

Operation: If cpcc false then (Dn − 1 → Dn; If Dn ≠ − 1 then PC + d → PC)

Assembler Syntax: cpDBcc Dn, <label>

Attributes: Size = (Word)

Description: If the specified coprocessor condition is met, execution continues with the next instruction. Otherwise, the low order word in the specified data register is decremented by one. If the result is equal to − 1, execution continues with the next instruction. If the result is not equal to − 1, execution continues at the location indicated by the current value of PC plus the sign extended 16-bit displacement. The value in the PC is the address of the displacement word. The coprocessor determines the specific condition from the condition word which follows the operation word.

Condition Codes: Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Cp-Id			0	0	1	0	0	1	Register		
0	0	0	0	0	0	0	0	0	0	Coprocessor Condition					
Optional Coprocessor Defined Extension Words															
Displacement															

Instruction Fields:

Cp-Id field — Identifies the coprocessor that is to process this operation.
Register field — Specifies the data register which is the counter.
Coprocessor Condition field — Specifies the coprocessor condition to be tested. This field is passed to the coprocessor, which provides directives to the main processor for processing this instruction.
Displacement field — Specifies the distance of the branch (in bytes).

cpGEN Coprocessor General Function cpGEN

Operation: Pass Command Word to Coprocessor

Assembler Syntax: cpGEN <parameters as defined by coprocessor>

Attributes: Unsized

Description: This instruction is the form used by coprocessors to specify the general data processing and movement operations. The coprocessor determines the specific operation from the command word which follows the operation word. Usually a coprocessor defines specific instances of this instruction to provide its instruction set.

Condition Codes: May be modified by coprocessor. Unchanged otherwise.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
										Effective Address					
1	1	1	1	Cp-Id			0	0	0	Mode			Register		
Coprocessor Command															
Optional Effective Address or Coprocessor Defined Extension Words															

Instruction Fields:

Cp-Id field — Identifies the coprocessor that is to process this operation.
Effective Address field — Specifies the location of any operand outside the coprocessor. The allowable addressing modes are determined by the operation to be performed.
Coprocessor Command field — Specifies the coprocessor operation to be performed. This word is passed to the coprocessor, which provides directives to the main processor for processing this instruction.

cpRESTORE Coprocessor Restore Functions (Privileged Instruction) cpRESTORE

Operation: Restore Internal State of Coprocessor

Assembler Syntax: cpRESTORE <ea>

Attributes: Unsized

Description: This instruction is used to restore the internal state of a coprocessor.

Condition Codes: Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Cp-Id			1	0	1	Effective Address Mode			Effective Address Register		

Instruction Field:

Cp-Id field — Identifies the coprocessor that is to be restored.

Effective Address field — Specifies the location where the internal state of the coprocessor is located. Only postincrement or control addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
−(An)	—	—
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

Programmer's Note: If the format word returned by the coprocessor indicates "come again", pending interrupts are not serviced.

cpSAVE Coprocessor Save Function (Privileged Instruction) cpSAVE

Operation: Save Internal State of Coprocessor

Assembler Syntax: cpSAVE <ea>

Attributes: Unsized

Description: This instruction is used to save the internal state of a coprocessor.

Condition Codes: Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Cp-Id			1	0	0	Effective Address Mode			Effective Address Register		

Instruction Fields:

Cp-Id field — Identifies the coprocessor that is to save its state.

Effective Address field — Specifies the location where the internal state of the coprocessor is to be saved. Only predecrement or alterable control addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An)+	—	—
−(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

cpScc — Set on Coprocessor Condition — cpScc

Operation: If cpcc true then 1s → Destination
else 0s → Destination

Assembler Syntax: cpScc <ea>

Attributes: Size = (Byte)

Description: The specified coprocessor condition code is tested; if the condition is true, the byte specified by the effective address is set to TRUE (all ones), otherwise that byte is set to FALSE (all zeros). The coprocessor determines the specific condition from the condition word which follows the operation word.

Condition Codes: Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Cp-Id			0	0	1	Effective Address Mode			Register		
0	0	0	0	0	0	0	0	0	0	Coprocessor Condition					
Optional Effective Address or Coprocessor Defined Extension Words															

Instruction Fields:

Cp-Id field — Identifies the coprocessor that is to process this operation.

Effective Address field — Specifies the destination location. Only data alterable addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
−(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

Coprocessor Condition field — Specifies the coprocessor condition to be tested. This field is passed to the coprocessor, which provides directives to the main processor for processing this instruction.

cpTRAPcc — Trap on Coprocessor Condition — cpTRAPcc

Operation: If cpcc true then TRAP

Assembler Syntax: cpTRAPcc
cpTRAPcc #<data>

Attributes: Unsized or Size = (Word, Long)

Description: If the selected coprocessor condition is true, the processor initiates exception processing. The vector number is generated to reference the cpTRAPcc exception vector, the stacked program counter is the address of the next instruction. If the selected condition is not true, no operation is performed, and execution continues with the next instruction. The coprocessor determines the specific condition from the condition word which follows the operation word. Following the condition word is a user-defined data operand specified as immediate data data, to be used by the trap handler.

Condition Codes: Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Cp-Id			0	0	1	1	1	1	Op-Mode		
0	0	0	0	0	0	0	0	0	0	Coprocessor Condition					
Optional Coprocessor Defined Extension Words															
Optional Word															
or Long Word Operand															

Instruction Fields:

Cp-Id field — Identifies the coprocessor that is to process this operation.

Op-Mode field — Selects the instruction form.

- 010—Instruction is followed by one operand word.
- 011—Instruction is followed by two operand words.
- 100—Instruction has no following operand words.

Coprocessor Condition field — Specifies the coprocessor condition to be tested. This field is passed to the coprocessor, which provides directives to the main processor for processing this instruction.

DBcc — Test Condition, Decrement, and Branch — DBcc

Operation: If condition false then ($Dn - 1 \rightarrow Dn$; If $Dn \neq -1$ then $PC + d \rightarrow PC$)

Assembler Syntax: DBcc Dn,<label>

Attributes: Size = (Word)

Description: This instruction is a looping primitive of three parameters: a condition, a counter (data register), and a displacement. The instruction first tests the condition to determine if the termination condition for the loop has been met, and if so, no operation is performed. If the termination condition is not true, the low order 16 bits of the counter data register are decremented by one. If the result is -1, the counter is exhausted and execution continues with the next instruction. If the result is not equal to -1, execution continues at the location indicated by the current value of the PC plus the sign-extended 16-bit displacement. The value in the PC is the current instruction location plus two.

"cc" may specify the following conditions:

CC	carry clear	0100	$\overline{C}$	LS	low or same	0011	$C+Z$
CS	carry set	0101	C	LT	less than	1101	$N \cdot \overline{V} + \overline{N} \cdot V$
EQ	equal	0111	Z	MI	minus	1011	N
F	never true	0001	0	NE	not equal	0110	$\overline{Z}$
GE	greater or equal	1100	$N \cdot V + \overline{N} \cdot \overline{V}$	PL	plus	1010	$\overline{N}$
GT	greater than	1110	$N \cdot V \cdot \overline{Z} + \overline{N} \cdot \overline{V} \cdot \overline{Z}$	T	always true	0000	1
HI	high	0010	$\overline{C} \cdot \overline{Z}$	VC	overflow clear	1000	$\overline{V}$
LE	less or equal	1111	$Z + N \cdot \overline{V} + \overline{N} \cdot V$	VS	overflow set	1001	V

Condition Codes: Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	Condition				1	1	0	0	1	Register		
Displacement															

Instruction Fields:

Condition field — One of the sixteen conditions discussed in description.
Register field — Specifies the data register which is the counter.
Displacement field — Specifies the distance of the branch (in bytes).

Notes:

1. The terminating condition is like that defined by the UNTIL loop constructs of high-level languages. For example: DBMI can be stated as "decrement and branch until minus".

DBcc — Test Condition, Decrement, and Branch — DBcc

2. Most assemblers accept DBRA for DBF for use when no condition is required for termination of a loop.
3. There are two basic ways of entering a loop: at the beginning or by branching to the trailing DBcc instruction. If a loop structure terminated with DBcc is entered at the beginning, the control index count must be one less than the number of loop executions desired. This count is useful for indexed addressing modes and dynamically specified bit operations. However, when entering a loop by branching directly to the trailing DBcc instruction, the control index should equal the loop execution count. In this case, if a zerc count occurs, the DBcc instruction will not branch, causing a complete bypass of the main loop.

DIVS DIVSL — Signed Divide — DIVS DIVSL

Operation: Destination/Source → Destination

Assembler Syntax:

DIVS.W<ea>,Dn	32/16 → 16r:16q
DIVS.L<ea>,Dq	32/32 → 32q
DIVS.L<ea>,Dr:Dq	64/32 → 32r:32q
DIVSL.L<ea>,Dr:Dq	32/32 → 32r:32q

Attributes: Size = (Word, Long)

Description: Divide the destination operand by the source and store the result in the destination. The operation is performed using signed arithmetic.

The instruction has a word form and three long forms. For the word form, the destination operand is a long word and the source operand is a word. The result is 32-bits, such that the quotient is in the lower word (least significant 16 bits) of the destination and the remainder is in the upper word (most significant 16 bits) of the destination. Note that the sign of the remainder is the same as the sign of the dividend.

For the first long form, the destination operand is a long word and the source operand is a long word. The result is a long quotient, and the remainder is discarded.

For the second long form, the destination operand is a quad word, contained in any two data registers, and the source operand is a long word. The result is a long word quotient and a long word remainder.

For the third long form, the destination operand is a long word and the source operand is a long word. The result is a long word quotient and a long word remainder.

Two special conditions may arise during the operation:
1. Division by zero causes a trap.
2. Overlow may be detected and set before completion of the instruction. If overflow is detected, the condition is flagged but the operands are unaffected.

Condition Codes:

X	N	Z	V	C
—	*	*	*	0

N Set if the quotient is negative. Cleared otherwise. Undefined if overflow or divide by zero.
Z Set if the quotient is zero. Cleared otherwise. Undefined if overflow or divide by zero.
V Set if division overflow is detected. Cleared otherwise.
C Always cleared.
X Not affected.

DIVS DIVSL — Signed Divide — DIVS DIVSL

Instruction Format (word form):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	Register Dn			1	1	1	Effective Address Mode			Register		

Instruction Fields:
Register field — Specifies any of the eight data registers. This field always specifies the destination operand.
Effective Address field — Specifies the source operand. Only data addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
−(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

Note: Overflow occurs if the quotient is larger than a 16-bit signed integer.

Instruction Format (long form):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	0	0	1	Effective Address Mode			Register		
0	Register Dq			1	Sz	0	0	0	0	0	0	0	Register Dr		

Instruction Fields:
Effective Address field — Specifies the source operand. Only data addressing modes are allowed as shown:

DIVS DIVSL — Signed Divide — DIVS DIVSL

Addr. Mode	Mode	Register	Addr. Mode	Mode	Register
Dn	000	reg. number:Dn	(xxx).W	111	000
An	—	—	(xxx).L	111	001
(An)	010	reg. number:An	#<data>	111	100
(An)+	011	reg. number:An			
–(An)	100	reg. number:An			
(d_{16},An)	101	reg. number:An	(d_{16},PC)	111	010
(d_8,An,Xn)	110	reg. number:An	(d_8,PC,Xn)	111	011
(bd,An,Xn)	110	reg. number:An	(bd,PC,Xn)	111	011
([bd,An,Xn],od)	110	reg. number:An	([bd,PC,Xn],od)	111	011
([bd,An],Xn,od)	110	reg. number:An	([bd,PC],Xn,od)	111	011

Register Dq field — Specifies a data register for the destination operand. The low order 32 bits of the dividend comes from this register, and the 32-bit quotient is loaded into this register.

Sz field — Selects a 32 or 64 bit division operation.
- 0—32-bit dividend is in Register Dq.
- 1—64-bit dividend is in Dr:Dq.

Register Dr field — After the division, the 32-bit remainder is loaded into this register. If Dr = Dq, only the quotient is returned. If Sz is 1, this field also specifies the data register in which the high order 32 bits of the dividend is located.

Note: Overflow occurs if the quotient is larger than a 32-bit signed integer.

DIVU DIVUL — Unsigned Divide — DIVU DIVUL

Operation: Destination/Source → Destination

Assembler Syntax:

DIVU.W<ea>,Dn	32/16 → 16r:16q
DIVU.L<ea>,Dq	32/32 → 32q
DIVU.L<ea>,Dr:Dq	64/32 → 32r:32q
DIVUL.L<ea>,Dr:Dq	32/32 → 32r:32q

Attributes: Size = (Word, Long)

Description: Divide the destination operand by the source and store the result in the destination. The operation is performed using unsigned arithmetic.

The instruction has a word from and three long forms. For the word form, the destination operand is a long word and the source operand is a word. The result is 32-bits, such that the quotient is in the lower word (least significant 16 bits) of the destination and the remainder is in the upper word (most significant 16 bits) of the destination. Note that the sign of the remainder is the same as the sign of the dividend.

For the first long form, the destination operand is a long word and the source operand is a long word. The result is a long word quotient, and the remainder is discarded.

For the second long form, the destination operand is a quad word, contained in any two data registers, and the source operand is a long word. The result is a long word quotient and a long word remainder.

For the third long form, the destination operand is a long word and the source operand is a long word. The result is a long word quotient and a long word remainder.

Two special conditions may arise:
1. Division by zero causes a trap.
2. Overlow may be detected and set before completion of the instruction. If overflow is detected, the condition is flagged but the operands are unaffected.

Condition Codes:

X	N	Z	V	C
—	*	*	*	0

- N Set if the quotient is negative. Cleared otherwise. Undefined if overflow or divide by zero.
- Z Set if the quotient is zero. Cleared otherwise. Undefined if overflow or divide by zero.
- V Set if division overflow is detected. Cleared otherwise.
- C Always cleared.
- X Not affected.

DIVU DIVUL — Unsigned Divide — DIVU DIVUL

Instruction Format (word form):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	Register Dn			0	1	1	Effective Address Mode			Effective Address Register		

Instruction Fields:

Register field — Specifies any of the eight data registers. This field always specifies the destination operand.

Effective Address field — Specifies the source operand. Only data addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
−(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

Note: Overflow occurs if the quotient is larger than a 16-bit unsigned integer.

Instruction Format (long form):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	0	0	1	Effective Address Mode			Effective Address Register		
0	Register Dq			0	Sz	0	0	0	0	0	0	0	Register Dr		

Instruction Fields:

Effective Address field — Specifies the source operand. Only data addressing modes are allowed as shown:

DIVU DIVUL — Unsigned Divide — DIVU DIVUL

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
−(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

Register Dq field — Specifies a data register for the destination operand. The low order 32 bits of the dividend come from this register, and the 32-bit quotient is loaded into this register.

Sz field — Selects a 32 or 64 bit division operation.
0—32-bit dividend is in Register Dq.
1—64-bit dividend is in Dr:Dq.

Register Dr field — After the division, the 32-bit remainder is loaded into this register. If Dr = Dq, only the quotient is returned. If Sz is 1, this field also specifies the data register in which the high order 32 bits of the dividend are located.

Note: Overflow occurs if the quotient is larger than a 32-bit unsigned integer.

EOR — Exclusive OR Logical — EOR

Operation: Source ⊕ Destination → Destination

Assembler Syntax: EOR Dn,<ea>

Attributes: Size = (Byte, Word, Long)

Description: Exclusive OR the source operand to the destination operand and store the result in the destination location. The size of the operation may be specified to be byte, word, or long. This operation is restricted to data registers as the source operand. The destination operand is specified in the effective address field.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

N Set if the most significant bit of the result is set. Cleared otherwise.
Z Set if the result is zero. Cleared otherwise.
V Always cleared.
C Always cleared.
X Not affected.

Instruction Format (word form):

15	14	13	12	11 10 9	8 7 6	5 4 3	2 1 0
						Effective Address	
1	0	1	1	Register Dn	Op-Mode	Mode	Register

Instruction Fields:
Register field — Specifies any of the eight data registers.
Op-Mode field —

Byte	**Word**	**Long**	**Operation**
100	101	110	<ea> ⊕ <Dx> → <ea>

Effective Address field — Specifies the destination operand. Only data alterable addressing modes are allowed as shown:

EOR — Exclusive OR Logical — EOR

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
−(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

Note: Memory to data register operations are not allowed. EORI is used when the source is immediate data. Most assemblers automatically make this distinction.

EORI — Exclusive OR Immediate — EORI

Operation: Immediate Data ⊕ Destination → Destination

Assembler Syntax: EORI #<data>,<ea>

Attributes: Size = (Byte, Word, Long)

Description: Exclusive OR the immediate data to the destination operand and store the result in the destination location. The size of the operation may be specified to be byte, word, or long. The immediate data matches the operation size.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

- N Set if the most significant bit of the result is set. Cleared otherwise.
- Z Set if the result is zero. Cleared otherwise.
- V Always cleared.
- C Always cleared.
- X Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	Size		Effective Address Mode			Effective Address Register		
Word Data (16 Bits)								Byte Data (8 Bits)							
Long Data (32 Bits, including Previous Word)															

Instruction Fields:

Size field — Specifies the size of the operation:
- 00—byte operation.
- 01—word operation.
- 10—long operation.

Effective Address field — Specifies the destination operand. Only data alterable addressing modes are allowed as shown:

EORI — Exclusive OR Immediate — EORI

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
–(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

Immediate field — (Data immediately following the instruction):
- If size = 00, then the data is the low order byte of the immediate word.
- If size = 01, then the data is the entire immediate word.
- If size = 10, then the data is next two immediate words.

EORI to CCR — Exclusive OR Immediate to Condition Code

Operation: Source ⊕ CCR → CCR

Assembler Syntax: EORI #<data>,CCR

Attributes: Size = (Byte)

Description: Exclusive OR the immediate operand with the condition codes and store the result in the low-order byte of the status register.

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

N Changed if bit 3 of immediate operand is one. Unchanged otherwise.
Z Changed if bit 2 of immediate operand is one. Unchanged otherwise.
V Changed if bit 1 of immediate operand is one. Unchanged otherwise.
C Changed if bit 0 of immediate operand is one. Unchanged otherwise.
X Changed if bit 4 of immediate operand is one. Unchanged otherwise.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	0	0	1	1	1	1	0	0
0	0	0	0	0	0	0	0	Byte Data (8 Bits)							

EORI to SR — Exclusive OR Immediate to the Status Register (Privileged Instruction)

Operation: If supervisor state
then Source ⊕ SR → SR
else TRAP

Assembler Syntax: EORI #<data>,SR

Attributes: Size = (Word)

Description: Exclusive OR the immediate operand with the contents of the status register and store the result in the status register. All bits of the status register are affected.

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

N Changed if bit 3 of immediate operand is one. Unchanged otherwise.
Z Changed if bit 2 of immediate operand is one. Unchanged otherwise.
V Changed if bit 1 of immediate operand is one. Unchanged otherwise.
C Changed if bit 0 of immediate operand is one. Unchanged otherwise.
X Changed if bit 4 of immediate operand is one. Unchanged otherwise.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	0	1	1	1	1	1	0	0
Word Data (16 Bits)															

EXG — Exchange Registers — EXG

Operation: Rx ↔ Ry

Assembler Syntax:
EXG Dx,Dy
EXG Ax,Ay
EXG Dx,Ay

Attributes: Size = (Long)

Description: Exchange the contents of two registers. This exchange is always a long (32 bit) operation. Exchange works in three modes:
1. Exchange data registers.
2. Exchange address registers.
3. Exchange a data register and an address register.

Condition Codes: Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	Register Rx			1	Op-Mode					Register Ry		

Instruction Fields:
Register Rx field — Specifies either a data register or an address register depending on the mode. If the exchange is between data and address registers, this field always specifies the data register.
Op-Mode field — Specifies whether exchanging:
01000—data registers.
01001—address registers.
10001—data register and address register.
Register Ry field — Specifies either a data register or an address register depending on the mode. If the exchange is between data and address registers, this field always specifies the address register.

EXT / EXTB — Sign Extend — EXT / EXTB

Operation: Destination Sign-extended → Destination

Assembler Syntax:

EXT.W Dn	extend byte to word
EXT.L Dn	extend word to long word
EXTB.L Dn	extend byte to long word

Attributes: Size = (Word, Long)

Description: Extend the sign bit of a data register from a byte to a word, from a word to a long word, or from a byte to a long word operand, depending on the size selected. If the operation is word, bit [7] of the designated data register is copied to bits [15:8] of that data register. If the operation is long, bit [15] of the designated data register is copied to bits [31:16] of the data register. The EXTB form copies bit [7] of the designated register to bits [31:8] of the data register.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

N Set if the result is negative. Cleared otherwise.
Z Set if the result is zero. Cleared otherwise.
V Always cleared.
C Always cleared.
X Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	Op-Mode			0	0	0	Register Dn		

Instruction Fields:
Op-Mode field — Specifies the size of the sign-extension operation:
010—Sign-extend low order byte of data register to word.
011—Sign-extend low order word of data register to long.
111—Sign-extend low order byte of data register to long.
Register field — Specifies the data register whose content is to be sign-extended.

ILLEGAL — Take Illegal Instruction Trap — ILLEGAL

Operation: SSP − 2 → SSP; Vector Offset → (SSP);
SSP − 4 → SSP; PC → (SSP);
SSP − 2 → SSP; SR → (SSP);
Illegal Instruction Vector Address → PC

Assembler Syntax: ILLEGAL

Attributes: Unsized

Description: This bit pattern causes an illegal instruction exception. All other illegal instruction bit patterns are reserved for future extension of the instruction set.

Condition Codes: Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	0	1	1	1	1	1	1	0	0

JMP — Jump — JMP

Operation: Destination Address → PC

Assembler Syntax: JMP <ea>

Attributes: Unsized

Description: Program execution continues at the effective address specified by the instruction. The address is specified by the control addressing modes.

Condition Codes: Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	1	1	Effective Address					
										Mode			Register		

Instruction Fields:

Effective Address field — Specifies the address of the next instruction. Only control addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An)+	—	—
−(An)	—	—
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

JSR — Jump to Subroutine — JSR

Operation: SP – 4 → SP; PC → (SP)
Destination Address → PC

Assembler Syntax: JSR <ea>

Attributes: Unsized

Description: The long word address of the instruction immediately following the JSR instruction is pushed onto the system stack. Program execution then continues at the address specified in the instruction.

Condition Codes: Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	1	0	Effective Address Mode			Effective Address Register		

Instruction Fields:
Effective Address field — Specifies the address of the next instruction. Only control addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An)+	—	—
–(An)	—	—
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

LEA — Load Effective Address — LEA

Operation: <ea> → An

Assembler Syntax: LEA <ea>,An

Attributes: Size = (Long)

Description: The effective address is loaded into the specified address register. All 32 bits of the address register are affected by this instruction.

Condition Codes: Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	Register An			1	1	1	Effective Address Mode			Effective Address Register		

Instruction Fields:
Register field — Specifies the address register which is to be loaded with the effective address.
Effective Address field — Specifies the address to be loaded into the address register. Only control addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An)+	—	—
–(An)	—	—
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

LINK — Link and Allocate — LINK

Operation: SP − 4 → SP; An → (SP);
SP → An; SP + d → SP

Assembler Syntax: LINK An, #<displacement>

Attributes: Size = (Word, Long)

Description: The current content of the specified address register is pushed onto the stack. After the push, the address register is loaded from the updated stack pointer. Finally, the displacement operand is added to the stack pointer. For word size operation, the displacement is the sign-extended word following the operation word. For long size operation, the displacement is the long word following the operation word. The content of the address register occupies one long word on the stack. A negative displacement is specified to allocate stack area.

Condition Codes: Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	1	0	Register		
Word Displacement															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	0	0	0	1	Register		
Long Displacement (High)															
Long Displacement (Low)															

Instruction Fields:
Register field — Specifies the address register through which the link is to be constructed.
Displacement field — Specifies the twos complement integer which is to be added to the stack pointer.

Note: LINK and UNLK can be used to maintain a linked list of local data and parameter areas on the stack for nested subroutine calls.

LSL, LSR — Logical Shift — LSL,LSR

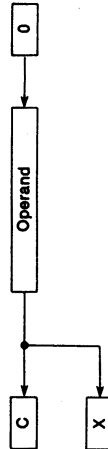
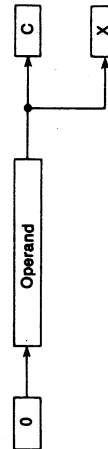
Operation: Destination Shifted by <count> → Destination

Assembler Syntax:
LSd Dx,Dy
LSd #<data>,Dy
LSd <ea>
where d is direction, L or R

Attributes: Size = (Byte, Word, Long)

Description: Shift the bits of the operand in the direction (L or R) specified. The carry bit receives the last bit shifted out of the operand. The shift count for the shifting of a register may be specified in two different ways:
1. Immediate — the shift count is specified in the instruction (shift range 1-8).
2. Register — the shift count is contained in a data register specified in the instruction (shift count modulo 64).

The size of the operation may be specified to be byte, word, or long. The content of memory may be shifted one bit only, and the operand size is restricted to a word.

For LSL, the operand is shifted left; the number of positions shifted is the shift count. Bits shifted out of the high order bit go to both the carry and the extend bits; zeroes are shifted into the low order bit.

For LSR, the operand is shifted right; the number of positions shifted is the shift count. Bits shifted out of the low order bit go to both the carry and the extend bits; zeroes are shifted into the high order bit.

— Continued —

LSL,LSR — Logical Shift — LSL,LSR

Condition Codes:

X	N	Z	V	C
*	*	*	0	*

N Set if the result is negative. Cleared otherwise.
Z Set if the result is zero. Cleared otherwise.
V Always cleared.
C Set according to the last bit shifted out of the operand. Cleared for a shift count of zero.
X Set according to the last bit shifted out of the operand. Unaffected for a shift count of zero.

Instruction Format (Register Shifts):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	Count/ Register			dr	Size		i/r	0	1	Register		

Instruction Field (Register Shifts):
Count/Register field —
If i/r = 0, the shift count is specified in this field. The values 0, 1-7 represent a range of 8, 1 to 7 respectively.
If i/r = 1, the shift count (modulo 64) is contained in the data register specified in this field.
dr field — Specifies the direction of the shift:
0—shift right.
1—shift left.
Size field — Specifies the size of the operation:
00—byte operation.
01—word operation.
10—long operation.
i/r field —
If i/r = 0, Specifies immediate shift count.
If i/r = 1, Specifies register shift count.
Register field — Specifies a data register whose content is to be shifted.

Instruction Format (Memory Shifts):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
										Effective Address					
1	1	1	0	0	0	1	dr	1	1	Mode			Register		

Instruction Fields (Memory Shifts):
dr field — Specifies the direction of the shift:
0—shift right.
1—shift left.
Effective Address field — Specifies the operand to be shifted. Only memory alterable addressing modes are allowed as shown:

LSL,LSR — Logical Shift — LSL,LSR

Addr. Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
−(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

MOVE — Move Data from Source to Destination — MOVE

Operation: Source → Destination

Assembler Syntax: MOVE <ea>,<ea>

Attributes: Size = (Byte, Word, Long)

Description: Move the content of the source to the destination location. The data is examined as it is moved, and the condition codes set accordingly. The size of the operation may be specified to be byte, word, or long.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

N Set if the result is negative. Cleared otherwise.
Z Set if the result is zero. Cleared otherwise.
V Always cleared.
C Always cleared.
X Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	Size		Destination Register			Destination Mode			Source Mode			Source Register		

Instruction Fields:

Size field — Specifies the size of the operand to be moved:
01—byte operation.
11—word operation.
10—long operation.

Destination Effective Address field — Specifies the destination location. Only data alterable addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	000	reg. number:An
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

MOVE — Move Data from Source to Destination — MOVE

Source Effective Address field — Specifies the source operand. All addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An*	001	reg. number:An
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*For byte size operation, address register direct is not allowed.

Notes: 1. MOVEA is used when the destination is an address register. Most assemblers automatically make this distinction.
2. MOVEQ can also be used for certain operations on data registers.

MOVEA — Move Address — MOVEA

Operation: Source → Destination

Assembler Syntax: MOVEA <ea>,An

Attributes: Size = (Word, Long)

Description: Move the content of the source to the destination address register. The size of the operation may be specified to be word or long. Word size source operands are sign extended to 32 bit quantities before the operation is done.

Condition Codes: Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	Size		Destination Register			0	0	1	Source Mode			Source Register		

Instruction Fields:

Size field — Specifies the size of the operand to be moved:
11—Word operation. The source operand is sign-extended to a long operand and all 32 bits are loaded into the address register.
10—Long operation.
Destination Register field — Specifies the destination address register.
Source Effective Address field — Specifies the location of source operand. All addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	001	reg. number:An
(An)	010	reg. number:An
(An)+	011	reg. number:An
−(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

MOVE from CCR — Move from the Condition Code Register — MOVE from CCR

Operation: CCR → Destination

Assembler Syntax: MOVE CCR,<ea>

Attributes: Size = (Word)

Description: The content of the status register is moved to the destination location. The source operand is a word, but only the low order byte contains the condition codes. The upper byte is all zeroes.

Condition Codes: Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	1	0	1	1	Effective Address Mode			Effective Address Register		

Instruction Fields:

Effective Address field — Specifies the destination location. Only data alterable addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
−(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

Note: MOVE from CCR is a word operation. ANDI, ORI, and EORI to CCR are byte operations.

MOVE to CCR — Move to Condition Codes — MOVE to CCR

Operation: Source → CCR

Assembler Syntax: MOVE <ea>,CCR

Attributes: Size = (Word)

Description: The content of the source operand is moved to the condition codes. The source operand is a word, but only the low order byte is used to update the condition codes. The upper byte is ignored.

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

- N Set the same as bit 3 of the source operand.
- Z Set the same as bit 2 of the source operand.
- V Set the same as bit 1 of the source operand.
- C Set the same as bit 0 of the source operand.
- X Set the same as bit 4 of the source operand.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	0	0	1	1	Effective Address Mode			Register		

Instruction Fields:

Effective Address field — Specifies the location of the source operand. Only data addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
−(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

Note: MOVE to CCR is a word operation. ANDI, ORI, and EORI to CCR are byte operations.

MOVE from SR — Move from the Status Register (Privileged Instruction) — MOVE from SR

Operation: If supervisor state
then SR → Destination
else TRAP

Assembler Syntax: MOVE SR,<ea>

Attributes: Size = (Word)

Description: The content of the status register is moved to the destination location. The operand size is a word.

Condition Codes: Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	1	1	Effective Address Mode			Register		

Instruction Fields:

Effective Address field — Specifies the destination location. Only data alterable addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
−(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

Note: Use the MOVE from CCR instruction to access only the condition codes.

MOVE to SR

Move to the Status Register (Privileged Instruction)

Operation: If supervisor state
then Source → SR
else TRAP

Assembler Syntax: MOVE <ea>,SR

Attributes: Size = (Word)

Description: The content of the source operand is moved to the status register. The source operand is a word and all bits of the status register are affected.

Condition Codes: Set according to the source operand.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	1	0	1	1	Effective Address Mode			Effective Address Register		

Instruction Fields:
Effective Address field — Specifies the location of the source operand. Only data addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
−(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

MOVE USP

Move User Stack Pointer (Privileged Instruction)

Operation: If supervisor state
then USP → An or An → USP
else TRAP

Assembler Syntax: MOVE USP,An
Move An,USP

Attributes: Size = (Long)

Description: The contents of the user stack pointer are transferred to or from the specified address register.

Condition Codes: Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	0	dr	Register		

Instruction Fields:
dr field — Specifies the direction of transfer:
0—transfer the address register to the USP.
1—transfer the USP to the address register.
Register field — Specifies the address register to or from which the user stack pointer is to be transferred.

MOVEC — Move Control Register (Privileged Instruction) — MOVEC

Operation: If supervisor state
then Rc → Rn or Rn → Rc
else TRAP

Assembler Syntax:
MOVEC Rc,Rn
MOVEC Rn,Rc

Attributes: Size = (Long)

Description: Copy the contents of the specified control register (Rc) to the specified general register or copy the contents of the specified general register to the specified control register. This is always a 32-bit transfer even though the control register may be implemented with fewer bits. Unimplemented bits are read as zeros.

Condition Codes: Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	1	0	1	dr
A/D	Register			Control Register											

Instruction Fields:
dr field — Specifies the direction of the transfer:
0—control register to general register.
1—general register to control register.
A/D field — Specifies the type of general register:
0—data register.
1—address register.
Register field — Specifies the register number.
Control Register field — Specifies the control register.

Hex	Control Register
000	Source Function Code (SFC) register.
001	Destination Function Code (DFC) register.
002	Cache Control Register (CACR).
800	User Stack Pointer (USP).
801	Vector Base Register (VBR).
802	Cache Address Register (CAAR).
803	Master Stack Pointer (MSP).
804	Interrupt Stack Pointer (ISP).

All other codes cause an illegal instruction exception.

MOVEM — Move Multiple Registers — MOVEM

Operation: Registers → Destination
Source → Registers

Assembler Syntax:
MOVEM register list,<ea>
MOVEM <ea>,register list

Attributes: Size = (Word, Long)

Description: Selected registers are transferred to or from consecutive memory locations starting at the location specified by the effective address. A register is transferred if the bit corresponding to that register is set in the mask field. The instruction selects how much of each register is transferred; either the entire long word can be moved or just the low order word. In the case of a word transfer to the registers, each word is sign-extended to 32 bits (including data registers) and the resulting long word loaded into the associated register.

MOVEM allows three forms of address modes: the control modes, the predecrement mode, or the postincrement mode. If the effective address is in one of the control modes, the registers are transferred starting at the specified address and up through higher addresses. The order of transfer is from data register 0 to data register 7, then from address register 0 to address register 7.

If the effective address is the predecrement mode, only a register to memory operation is allowed. The registers are stored starting at the specified address minus the operand length (2 or 4) and down through lower addresses. The order of storing is from address register 7 to address register 0, then from data register 7 to data register 0. The decremented address register is updated to contain the address of the last word stored.

If the effective address is the postincrement mode, only a memory to register operation is allowed. The registers are loaded starting at the specified address and up through higher addresses. The order of loading is the same as for the control mode addressing. The incremented address register is updated to contain the address of the last word loaded plus the operand length (2 or 4).

Condition Codes: Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	dr	0	0	1	Sz	Effective Address					
										Mode			Register		
Register List Mask															

— Continued —

MOVEM — Move Multiple Registers — MOVEM

Instruction Fields:

dr field — Specifies the direction of the transfer:
0—register to memory.
1—memory to register.

Sz field — Specifies the size of the registers being transferred:
0—word transfer.
1—long transfer.

Effective Address field — Specifies the memory address to or from which the registers are to be moved.

For register to memory transfers, only control alterable addressing modes or the predecrement addressing mode are allowed as shown:

Addr. Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An)+	011	—
−(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

For memory to register transfers, only control addressing modes or the post-increment addressing mode are allowed as shown:

Addr. Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
−(An)	—	—
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

Register List Mask field — Specifies which registers are to be transferred. The low order bit corresponds to the first register to be transferred; the high bit corresponds to the last register to be transferred. Thus, both for control modes and for the postincrement mode addresses, the mask correspondence is

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A7	A6	A5	A4	A3	A2	A1	A0	D7	D6	D5	D4	D3	D2	D1	D0

MOVEM — Move Multiple Registers — MOVEM

while for the predecrement mode addresses, the mask correspondence is

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D0	D1	D2	D3	D4	D5	D6	D7	A0	A1	A2	A3	A4	A5	A6	A7

Note: An extra read bus cycle occurs for memory operands. This accesses an operand at one address higher than the last register image required.

MOVEP — Move Peripheral Data — MOVEP

Operation: Source → Destination

Assembler Syntax: MOVEP Dx,(d,Ay)
MOVEP (d,Ay)Dx

Attributes: Size = (Word, Long)

Description: Data is transferred between a data register and alternate bytes of memory, starting at the location specified and incrementing by two. The high order byte of the data register is transferred first and the low order byte is transferred last. The memory address is specified using the address register indirect plus 16-bit displacement addressing mode. This instruction is designed to work with 8-bit peripherals on a 16-bit data bus. If the address is even, all the transfers are made on the high order half of the data bus; if the address is odd, all the transfers are made on the low order half of the data bus. On an 8- or 32-bit bus, the instruction still accesses every other byte.

Example: Long transfer to/from an even address.

Byte organization in register

31–24	23–16	15–8	7–0
Hi-Order	Mid-Upper	Mid-Lower	Low-Order

Byte organization in memory (low address at top)

15–8	7–0
Hi-Order	
Mid-Upper	
Mid-Lower	
Low-Order	

Example: Word transfer to/from an odd address.

Byte organization in register

31–24	23–16	15–8	7–0
		Hi-Order	Low-Order

Byte organization in memory (low address at top)

15–8	7–0
	Hi-Order
	Low-Order

— Continued —

MOVEP — Move Peripheral Data — MOVEP

Condition Codes: Not affected.

Instruction Format:

15	14	13	12	11 10 9	8 7 6	5	4	3	2 1 0
0	0	0	0	Data Register	Op-Mode	0	0	1	Address Register
Displacement									

Instruction Fields:

Data Register field — Specifies the data register to or from which the data is to be transferred.

Op-Mode field — Specifies the direction and size of the operation:
- 100—transfer word from memory to register.
- 101—transfer long from memory to register.
- 110—transfer word from register to memory.
- 111—transfer long from register to memory.

Address Register field — Specifies the address register which is used in the address register indirect plus displacement addressing mode.

Displacement field — Specifies the displacement which is used in calculating the operand address.

MOVEQ — Move Quick — MOVEQ

Operation: Immediate Data → Destination

Assembler Syntax: MOVEQ #<data>,Dn

Attributes: Size = (Long)

Description: Move immediate data to a data register. The data is contained in an 8-bit field within the operation word. The data is sign-extended to a long operand and all 32 bits are transferred to the data register.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

N Set if the result is negative. Cleared otherwise.
Z Set if the result is zero. Cleared otherwise.
V Always cleared.
C Always cleared.
X Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	Register			0	Data							

Instruction Fields:
Register field — Specifies the data register to be loaded.
Data field — 8 bits of data which are sign extended to a long operand.

MOVES — Move Address Space (Privileged Instruction) — MOVES

Operation: If supervisor state
then Rn → Destination [DFC] or Source [SFC] → Rn
else TRAP

Assembler Syntax: MOVES Rn,<ea>
MOVES <ea>,Rn

Attributes: Size = (Byte, Word, Long)

Description: Move the byte, word, or long operand from the specified general register to a location within the address space specified by the destination function code (DFC) register. Or, move the byte, word, or long operand from a location within the address space specified by the source function code (SFC) register to the specified general register.

If the destination is a data register, the source operand replaces the corresponding low-order bits of that data register. If the destination is an address register, the source operand is sign-extended to 32 bits and then loaded into that address register.

Condition Codes: Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
										Effective Address					
										Mode			Register		
0	0	0	0	1	1	1	0	Size							
A/D	Register			dr	0	0	0	0	0	0	0	0	0	0	0

Instruction Fields:
Size field — Specifies the size of the operation:
00—byte operation.
01—word operation.
10—long operation.
Effective Address field — Specifies the source or destination location within the alternate address space. Only alterable memory addressing modes are allowed as shown:

— Continued —

MOVES — Move Address Space (Privileged Instruction) — MOVES

Addr. Mode	Mode	Register	Addr. Mode	Mode	Register
Dn	—	—	(xxx).W	111	000
An	—	—	(xxx).L	111	001
(An)	010	reg. number:An	#<data>	—	—
(An)+	011	reg. number:An			
-(An)	100	reg. number:An			
(d_{16},An)	101	reg. number:An	(d_{16},PC)	—	—
(d_8,An,Xn)	110	reg. number:An	(d_8,PC,Xn)	—	—
(bd,An,Xn)	110	reg. number:An	(bd,PC,Xn)	—	—
([bd,An,Xn],od)	110	reg. number:An	([bd,PC,Xn],od)	—	—
([bd,An],Xn,od)	110	reg. number:An	([bd,PC],Xn,od)	—	—

A/D field — Specifies the type of general register:
0—data register.
1—address register.
Register field — Specifies the register number.
dr field — Specifies the direction of the transfer:
0—from <ea> to general register.
1—from general register to <ea>.

MOVES.x An(An)+
or
MOVES.x An, -(An)

where An is the same address register for both source and destination and is an undefined operation. The value stored in memory is undefined.

NOTE

On the MC68010 and MC68020 implementations, the value stored is the increment or the decrement value of An. This implementation may not appear on future devices.

MULS — Signed Multiply — MULS

Operation: Source*Destination → Destination

Assembler Syntax:
MULS.W<ea>,Dn 16 x 16 → 32
MULS.L<ea>,Dl 32 x 32 → 32
MULS.L<ea>,Dh:Dl 32 x 32 → 64

Attributes: Size = (Word, Long)

Description: Multiply two signed operands yielding a signed result. The operation is performed using signed arithmetic.

The instruction has a word form and a long form. For the word form, the multiplier and multiplicand are both word operands and the result is long word operand. A register operand is taken from the low order word, the upper word is unused. All 32 bits of the product are saved in the destination data register.

For the long form, the multiplier and multiplicand are both long word operands and the result is either a long word or a quad word. The long word result is the low order 32 bits of the quad word result.

Condition Codes:

X	N	Z	V	C
—	*	*	*	0

N Set if the result is negative. Cleared otherwise.
Z Set if the result is zero. Cleared otherwise.
V Set if overflow. Cleared otherwise.
C Always cleared.
X Not affected.

Note: Overflow (V = 1) can occur only in the case of multiplying 32-bit operands to yield a 32-bit result. Overflow occurs if the high-order 32 bits of the quad word product are not the sign-extension of the low order 32 bits.

Instruction Format (word form):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	Register Dn			1	1	1	Effective Address Mode			Effective Address Register		

Instruction Fields:
Register field — Specifies one of the data registers. This field always specifies the destination.
Effective Address field — Specifies the source operand. Only data addressing modes are allowed as shown:

MULS — Signed Multiply — MULS

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
−(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

Instruction Format (long form):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	0	0	0	Effective Address Mode			Register		
0	Register Dl			1	Sz	0	0	0	0	0	0	0	Register Dh		

Instruction Fields:

Effective Address field — Specifies the source operand. Only data addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
−(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

Register Dl field — Specifies a data register for the destination operand. The 32-bit multiplicand comes from this register, and the low order 32 bits of the product is loaded into this register.

Sz field — Selects a 32- or 64-bit product result.
0—32-bit product to be returned to Register Dl.
1—64-bit product to be returned to Dh:Dl.

Register Dh field — If Sz is 1, specifies the data register into which the high order 32 bits of the product is loaded. If Dh = Dl and Sz is 1, the results of the operation are undefined. Otherwise, this field is unused.

MULU — Unsigned Multiply — MULU

Operation: Source*Destination → Destination

Assembler Syntax:

MULU.W<ea>,Dn	16 × 16 → 32
MULU.L<ea>,Dl	32 × 32 → 32
MULU.L<ea>,Dh:Dl	32 × 32 → 64

Attributes: Size = (Word, Long)

Description: Multiply two unsigned operands yielding a unsigned result. The operation is performed using unsigned arithmetic.

The instruction has a word form and a long form. For the word form, the multiplier and multiplicand are both word operands and the result is a long word operand. A register operand is taken from the low order word, the upper word is unused. All 32 bits of the product are saved in the destination data register.

For the long form, the multiplier and multiplicand are both long word operands and the result is either a long word or a quad word. The long word result is the low order 32 bits of the quad word result.

Condition Codes:

X	N	Z	V	C
—	*	*	*	0

N Set if the result is negative. Cleared otherwise.
Z Set if the result is zero. Cleared otherwise.
V Set if overflow. Cleared otherwise.
C Always cleared.
X Not affected.

Note: Overflow (V = 1) can occur only in the case of multiplying 32-bit operands to yield a 32-bit result. Overflow occurs if the high-order 32 bits of the quad word product are non-zero.

Instruction Format (word form):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	Register Dn			0	1	1	Effective Address Mode			Register		

Instruction Fields:

Register field — Specifies one of the data registers. This field always specifies the destination.

Effective Address field — Specifies the source operand. Only data addressing modes are allowed as shown:

MULU — Unsigned Multiply — MULU

Addr. Mode	Mode	Register	Addr. Mode	Mode	Register
Dn	000	reg. number:Dn	(xxx).W	111	000
An	—	—	(xxx).L	111	001
(An)	010	reg. number:An	#<data>	111	100
(An)+	011	reg. number:An			
−(An)	100	reg. number:An			
(d_{16},An)	101	reg. number:An	(d_{16},PC)	111	010
(d_8,An,Xn)	110	reg. number:An	(d_8,PC,Xn)	111	011
(bd,An,Xn)	110	reg. number:An	(bd,PC,Xn)	111	011
([bd,An,Xn],od)	110	reg. number:An	([bd,PC,Xn],od)	111	011
([bd,An],Xn,od)	110	reg. number:An	([bd,PC],Xn,od)	111	011

Instruction Format (long form):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	0	0	0	Effective Address Mode			Effective Address Register		
0	Register Dl			0	Sz	0	0	0	0	0	0	0	Register Dh		

Instruction Fields:

Effective Address field — Specifies the source operand. Only data addressing modes are allowed as shown:

Addr. Mode	Mode	Register	Addr. Mode	Mode	Register
Dn	000	reg. number:Dn	(xxx).W	111	000
An	—	—	(xxx).L	111	001
(An)	010	reg. number:An	#<data>	111	100
(An)+	011	reg. number:An			
−(An)	100	reg. number:An			
(d_{16},An)	101	reg. number:An	(d_{16},PC)	111	010
(d_8,An,Xn)	110	reg. number:An	(d_8,PC,Xn)	111	011
(bd,An,Xn)	110	reg. number:An	(bd,PC,Xn)	111	011
([bd,An,Xn],od)	110	reg. number:An	([bd,PC,Xn],od)	111	011
([bd,An],Xn,od)	110	reg. number:An	([bd,PC],Xn,od)	111	011

Register Dl field — Specifies a data register for the destination operand. The 32-bit multiplicand comes from this register, and the low order 32 bits of the product are loaded into this register.

Sz field — Selects a 32- or 64-bit product result.
0—32-bit product to be returned to Register Dl.
1—64-bit product to be returned to Dh:Dl.

Register Dh field — If Sz is 1, specifies the data register into which the high order 32 bits of the product are loaded. If Dh = Dl and Sz is 1, the results of the operation are undefined. Otherwise, this field is unused.

NBCD — Negate Decimal with Extend — NBCD

Operation: 0 − $Destination_{10}$ − X → Destination

Assembler Syntax: NBCD <ea>

Attributes: Size = (Byte)

Description: The operand addressed as the destination and the extend bit are subtracted from zero. The operation is performed using decimal arithmetic. The result is saved in the destination location. This instruction produces the tens complement of the destination if the extend bit is clear, the nines complement if the extend bit is set. This is a byte operation only.

Condition Codes:

X	N	Z	V	C
*	U	*	U	*

N Undefined.
Z Cleared if the result is non-zero. Unchanged otherwise.
V Undefined.
C Set if a borrow (decimal) was generated. Cleared otherwise.
X Set the same as the carry bit.

NOTE

Normally the Z condition code bit is set via programming before the start of an operation. This allows successful tests for zero results upon completion of multiple precision operations.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	0	Effective Address Mode			Effective Address Register		

Instruction Fields:

Effective Address field — Specifies the destination operand. Only data alterable addressing modes are allowed as shown:

NBCD — Negate Decimal with Extend — NBCD

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
−(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

NEG — Negate — NEG

Operation: 0 − Destination → Destination

Assembler Syntax: NEG <ea>

Attributes: Size = (Byte, Word, Long)

Description: The operand addressed as the destination is subtracted from zero. The result is stored in the destination location. The size of the operation may be specified to be byte, word, or long.

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

N Set if the result is negative. Cleared otherwise.
Z Set if the result is zero. Cleared otherwise.
V Set if an overflow is generated. Cleared otherwise.
C Cleared if the result is zero. Set otherwise.
X Set the same as the carry bit.

Instruction Format:

15	14	13	12	11	10	9	8	7 6	5 4 3	2 1 0
									Effective Address	
0	1	0	0	0	1	0	0	Size	Mode	Register

Instruction Fields:

Size field — Specifies the size of the operation.
00—byte operation.
01—word operation.
10—long operation.

Effective Address field — Specifies the destination operand. Only data alterable addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
−(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

NEGX — Negate with Extend — NEGX

Operation: 0 – (Destination) – X → Destination

Assembler Syntax: NEGX <ea>

Attributes: Size = (Byte, Word, Long)

Description: The operand addressed as the destination and the extend bit are subtracted from zero. The result is stored in the destination location. The size of the operation may be specified to be byte, word, or long.

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

N Set if the result is negative. Cleared otherwise.
Z Cleared if the result is non-zero. Unchanged otherwise.
V Set if an overflow is generated. Cleared otherwise.
C Set if a borrow is generated. Cleared otherwise.
X Set the same as the carry bit.

NOTE

Normally the Z condition code bit is set via programming before the start of an operation. This allows successful tests for zero results upon completion of multiple-precision operations.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	Size		Effective Address Mode			Effective Address Register		

Instruction Fields:

Size field — Specifies the size of the operation.
00—byte operation.
01—word operation.
10—long operation.

Effective Address field — Specifies the destination operand. Only data alterable addressing modes are allowed as shown:

NEGX — Negate with Extend — NEGX

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
–(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

NOP — No Operation — NOP

Operation: None

Assembler Syntax: NOP

Attributes: Unsized

Description: No operation occurs. The processor state, other than the program counter, is unaffected. Execution continues with the instruction following the NOP instruction. The NOP instruction does not complete execution until all pending bus cycles are completed. This allows synchronization of the pipeline to be accomplished, and prevents instruction overlap.

Condition Codes: Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	0	1

NOT — Logical Complement — NOT

Operation: ~ Destination → Destination

Assembler Syntax: NOT <ea>

Attributes: Size = (Byte, Word, Long)

Description: The ones complements of the destination operand is taken and the result is stored in the destination location. The size of the operation may be specified to be byte, word, or long.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

N Set if the result is negative. Cleared otherwise.
Z Set if the result is zero. Cleared otherwise.
V Always cleared.
C Always cleared.
X Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	1	0	Size		Effective Address Mode			Register		

Instruction Fields:

Size field — Specifies the size of the operation.
00—byte operation.
01—word operation.
10—long operation.

Effective Address field — Specifies the destination operand. Only data alterable addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
−(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

OR — Inclusive OR Logical — OR

Operation: Source v Destination → Destination

Assembler Syntax: OR <ea>,Dn
OR Dn,<ea>

Attributes: Size = (Byte, Word, Long)

Description: Inclusive OR the source operand to the destination operand and store the result in the destination location. The size of the operation may be specified to be byte, word, or long. The contents of an address register may not be used as an operand.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

N Set if the most significant bit of the result is set. Cleared otherwise.
Z Set if the result is zero. Cleared otherwise.
V Always cleared.
C Always cleared.
X Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	Register			Op-Mode			Effective Address					
										Mode			Register		

Instruction Fields:
Register field — Specifies any of the eight data registers.
Op-Mode field —

Byte	Word	Long	Operation
000	001	010	(<ea>)v(<Dn>)→<Dn>
100	101	110	(<Dn>)v(<ea>)→<ea>

Effective Address field —
If the location specified is a source operand then only data addressing modes are allowed as shown:

OR — Inclusive OR Logical — OR

Addr. Mode	Mode	Register	Addr. Mode	Mode	Register
Dn	000	reg. number:Dn	(xxx).W	111	000
An	—	—	(xxx).L	111	001
(An)	010	reg. number:An	#<data>	111	100
(An)+	011	reg. number:An			
−(An)	100	reg. number:An			
(d_{16},An)	101	reg. number:An	(d_{16},PC)	111	010
(d_8,An,Xn)	110	reg. number:An	(d_8,PC,Xn)	111	011
(bd,An,Xn)	110	reg. number:An	(bd,PC,Xn)	111	011
([bd,An,Xn],od)	110	reg. number:An	([bd,PC,Xn],od)	111	011
([bd,An],Xn,od)	110	reg. number:An	([bd,PC],Xn,od)	111	011

If the location specified is a destination operand then only memory alterable addressing modes are allowed as shown:

Addr. Mode	Mode	Register	Addr. Mode	Mode	Register
Dn	—	—	(xxx).W	111	000
An	—	—	(xxx).L	111	001
(An)	010	reg. number:An	#<data>	—	—
(An)+	011	reg. number:An			
−(An)	100	reg. number:An			
(d_{16},An)	101	reg. number:An	(d_{16},PC)	—	—
(d_8,An,Xn)	110	reg. number:An	(d_8,PC,Xn)	—	—
(bd,An,Xn)	110	reg. number:An	(bd,PC,Xn)	—	—
([bd,An,Xn],od)	110	reg. number:An	([bd,PC,Xn],od)	—	—
([bd,An],Xn,od)	110	reg. number:An	([bd,PC],Xn,od)	—	—

Notes: 1. If the destination is a data register, then it cannot be specified by using the destination <ea> mode, but must use the destination Dn mode instead.
2. ORI is used when the source is immediate data. Most assemblers automatically make this distinction.

ORI — Inclusive OR Immediate — ORI

Operation: Immediate Data v Destination → Destination

Assembler Syntax: ORI #<data>,<ea>

Attributes: Size = (Byte, Word, Long)

Description: Inclusive OR the immediate data to the destination operand and store the result in the destination location. The size of the operation may be specified to be byte, word, or long. The size of the immediate data matches the operation size.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

N Set if the most significant bit of the result is set. Cleared otherwise.
Z Set if the result is zero. Cleared otherwise.
V Always cleared.
C Always cleared.
X Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	Size		Effective Address Mode			Effective Address Register		
Word Data								Byte Data							
Long Data															

Instruction Fields:
Size field — Specifies the size of the operation.
00—byte operation.
01—word operation.
10—long operation.
Effective Address field — Specifies the destination operand. Only data alterable addressing modes are allowed as shown:

ORI — Inclusive OR Immediate — ORI

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
−(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

Immediate field — (Data immediately following the instruction):
If size = 00, then the data is the low order byte of the immediate word.
If size = 01, then the data is the entire immediate word.
If size = 10, then the data is the next two immediate words.

ORI to CCR — Inclusive OR Immediate to Condition Codes

Operation: Source v CCR → CCR

Assembler Syntax: ORI #<data>,CCR

Attributes: Size = (Byte)

Description: Inclusive OR the immediate operand with the condition codes and store the result in the low-order byte of the status register.

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

N Set if bit 3 of immediate operand is one. Unchanged otherwise.
Z Set if bit 2 of immediate operand is one. Unchanged otherwise.
V Set if bit 1 of immediate operand is one. Unchanged otherwise.
C Set if bit 0 of immediate operand is one. Unchanged otherwise.
X Set if bit 4 of immediate operand is one. Unchanged otherwise.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0
0	0	0	0	0	0	0	0	Byte Data (8 Bits)							

ORI to SR — Inclusive OR Immediate to the Status Register (Privileged Instruction)

Operation: If supervisor state
then Source v SR → SR
else TRAP

Assembler Syntax: ORI #<data>,SR

Attributes: Size = (Word)

Description: Inclusive OR the immediate operand with the contents of the status register and store the result in the status register. All bits of the status register are affected.

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

N Set if bit 3 of immediate operand is one. Unchanged otherwise.
Z Set if bit 2 of immediate operand is one. Unchanged otherwise.
V Set if bit 1 of immediate operand is one. Unchanged otherwise.
C Set if bit 0 of immediate operand is one. Unchanged otherwise.
X Set if bit 4 of immediate operand is one. Unchanged otherwise.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0
Word Data (16 Bits)															

PACK — Pack — PACK

Operation: Source (Unpacked BCD) + adjustment → Destination (Packed BCD)

Assembler Syntax: PACK −(Ax), −(Ay),#<adjustment>
PACK Dx,Dy,#<adjustment>

Attributes: Unsized

Description: The low four bits of each of two bytes are adjusted and packed into a single byte.

When both operands are data registers, the adjustment is added to the value contained in the source register. Bits [11:8] and [3:0] of the intermediate result are concatenated and placed in bits [7:0] of the destination register. The remainder of the destination register is unaffected.

Source:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Dx	x	x	x	x	a	b	c	d	x	x	x	x	e	f	g	h

Add Adjustment Word:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
16-Bit Extension															

Resulting in:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
x'	x'	x'	x'	a'	b'	c'	d'	x'	x'	x'	x'	e'	f'	g'	h'

Destination:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Dy	u	u	u	u	u	u	u	u	a'	b'	c'	d'	e'	f'	g'	h'

When the addressing mode specified is predecrement, two bytes from the source are fetched, adjusted, and concatenated. The extension word is added to the concatenated bytes. Bits [3:0] of each byte are extracted. These eight bits are concatenated to form a new byte which is then written to the destination.

Source:

	7	6	5	4	3	2	1	0
	x	x	x	x	a	b	c	d
	x	x	x	x	e	f	g	h
(Ax)								

Concatenated Word:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
x	x	x	x	a	b	c	d	x	x	x	x	e	f	g	h

PACK — Pack — PACK

Add Adjustment Word:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
16-Bit Extension															

Destination:

	7	6	5	4	3	2	1	0
	a'	b'	c'	d'	e'	f'	g'	h'
(Ay)								

Condition Codes: Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	Register Dy/Ay			1	0	1	0	0	R/M	Register Dx/Ax		
Adjustment															

Instruction Fields:

Register Dy/Ay field — Specifies the destination register.
If R/M = 0, specifies a data register.
If R/M = 1, specifies an address register for the predecrement addressing mode.

R/M field — Specifies the operand addressing mode.
0—The operation is data register to data register.
1—The operation is memory to memory.

Register Dx/Ax field — Specifies the source register.
If R/M = 0, specifies a data register.
If R/M = 1, specifies an address register for the predecrement addressing mode.

Adjustment field — Immediate data word which is added to the source operand. Appropriate constants can be used to translate from ASCII or EBCDIC to BCD.

PEA — Push Effective Address — PEA

Operation: SP − 4 → SP; EA → (SP)

Assembler Syntax: PEA <ea>

Attributes: Size = (Long)

Description: The effective address is computed and pushed onto the stack. A long word address is pushed onto the stack.

Condition Codes: Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	1	Effective Address Mode			Effective Address Register		

Instruction Fields:

Effective Address field — Specifies the address to be pushed onto the stack. Only control addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An)+	—	—
−(An)	—	—
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

RESET — Reset External Devices (Privileged Instruction) — RESET

Operation: If supervisor state
then Assert RESET Line
else TRAP

Assembler Syntax: RESET

Attributes: Unsized

Description: The reset line is asserted, causing all external devices to be reset. The processor state, other than the program counter, is unaffected and execution continues with the next instruction.

Condition Codes: Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	0	0

ROL / ROR — Rotate (Without Extend)

Operation: Destination Rotated by <count> → Destination

Assembler Syntax:
ROd Dx,Dy
ROd #<data>,Dy
ROd <ea>
where d is direction, L or R

Attributes: Size = (Byte, Word, Long)

Description: Rotate the bits of the operand in the direction (L or R) specified. The extend bit is not included in the rotation. The rotate count for the rotation of a register may be specified in two different ways:

1. Immediate — the rotate count is specified in the instruction (rotate range, 1-8).
2. Register — the rotate count is contained in a data register specified in the instruction.

The size of the operation may be specified to be byte, word, or long. The content of memory may be rotated by one bit only and the operand size is restricted to a word.

For ROL, the operand is rotated left; the number of positions rotated is the rotate count. Bits rotated out of the high order bit go to both the carry bit and back into the low order bit. The extend bit is not modified or used.

ROL: [C] ← [Operand] (high order bit goes to C and back into the low order bit)

For ROR, the operand is rotated right; the number of positions rotated is the rotate count. Bits shifted out of the low order bit go to both the carry bit and back into high order bit. The extend bit is not modified or used.

ROR: [Operand] → [C] (low order bit goes to C and back into the high order bit)

— Continued —

ROL / ROR — Rotate (Without Extend)

Condition Codes:

X	N	Z	V	C
—	*	*	0	*

N Set if the most significant bit of the result is set. Cleared otherwise.
Z Set if the result is zero. Cleared otherwise.
V Always cleared.
C Set according to the last bit rotated out of the operand. Cleared for a rotate count of zero.
X Not affected.

Instruction Format (Register Rotate):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	Rotate/Register			dr	Size		i/r	1	1	Register		

Instruction Fields (Register Rotate):
Rotate/Register field —
If i/r = 0, the rotate count is specified in this field. The values 0, 1-7 represent a range of 8, 1 to 7 respectively.
If i/r = 1, the rotate count (modulo 64) is contained in the data register specified in this field.
dr field — Specifies the direction of the rotate:
0—rotate right.
1—rotate left.
Size field — Specifies the size of the operation:
00—byte operation.
01—word operation.
10—long operation.
i/r field —
If i/r = 0, specifies immediate rotate count.
If i/r = 1, specifies register rotate count.
Register field — Specifies a data register whose content is to be rotated.

Instruction Format (Memory Rotate):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
										Effective Address					
1	1	1	0	0	1	1	dr	1	1	Mode			Register		

ROL / ROR — Rotate (without Extend) — ROL / ROR

Instruction Fields (Memory Rotate):

dr field — Specifies the direction of the rotate:
- 0—rotate right.
- 1—rotate left.

Effective Address field — Specifies the operand to be rotated. Only memory alterable addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
*		
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

ROXL / ROXR — Rotate with Extend — ROXL / ROXR

Operation: Destination Rotated with X by <count> → Destination

Assembler Syntax:
ROXd Dx,Dy
ROXd #<data>,Dy
ROXd <ea>

Attributes: Size = (Byte, Word, Long)

Description: Rotate the bits of the destination operand in the direction specified. The extend bit (X) is included in the rotation. The rotate count for the rotation of a register may be specified in two different ways:
1. Immediate — the rotate count is specified in the instruction (rotate range, 1-8).
2. Register — the rotate count (modulo 64) is contained in a data register specified in the instruction.

The size of the operation may be specified to be byte, word, or long. The content of memory may be rotated one bit only and the operand size is restricted to a word.

For ROXL, the operand is rotated left; the number of positions rotated is the rotate count. Bits rotated out of the high order bit go to both the carry and extend bits; the previous value of the extend bit is rotated into the low order bit.

ROXL: C ← Operand ← X

For ROXR, the operand is rotated right; the number of positions shifted is the rotate count. Bits rotated out of the low order bit go to both the carry and extend bits; the previous value of the extend bit is rotated into the high order bit.

ROXR: X → Operand → C

Condition Codes:

X	N	Z	V	C
*	*	*	0	*

- N Set if the most significant bit of the result is set. Cleared otherwise.
- Z Set if the result is zero. Cleared otherwise.
- V Always cleared.
- C Set according to the last bit rotated out of the operand. Set to the value of the extend bit for a rotate count of zero.
- X Set according to the last bit rotated out of the operand. Unaffected for a rotate count of zero.

ROXL ROXR — Rotate with Extend

Instruction Format (Register Rotate):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	Rotate/ Register			dr	Size		i/r	1	0	Register		

Instruction Fields (Register Rotate):

Count/Register field —

If i/r = 0, the rotate count is specified in this field. The values 0, 1-7 represent a range of 8, 1 to 7 respectively.

If i/r = 1, the rotate count (modulo 64) is contained in the data register specified in this field.

dr field — Specifies the direction of the rotate:
- 0—rotate right.
- 1—rotate left.

Size field — Specifies the size of the operation:
- 00—byte operation.
- 01—word operation.
- 10—long operation.

i/r field —
- If i/r = 0, specifies immediate rotate count.
- If i/r = 1, specifies register rotate count.

Register field — Specifies a data register wnose content is to be rotated.

Instruction Format (Memory Rotate):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	1	0	dr	1	1	Effective Address Mode			Register		

Instruction Fields (Memory Rotate):

dr field — Specifies the direction of the rotate:
- 0—rotate right.
- 1—rotate left.

Effective Address field — Specifies the operand to be rotated. Only memory alterable addressing modes are allowed as shown:

ROXL ROXR — Rotate with Extend

Addr. Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
-(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

RTD — Return and Deallocate Parameters

Operation: (SP) → PC; SP + 4 + d → SP

Assembler Syntax: RTD #<displacement>

Attributes: Unsized

Description: The program counter is pulled from the stack. The previous program counter value is lost. After the program counter is read from the stack, the displacement value (16 bits) is sign-extended to 32 bits and added to the stack pointer.

Condition Codes: Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	0	0
Displacement															

Instruction Field:

Displacement field — Specifies the twos complement integer which is to be sign-extended and added to the stack pointer.

RTE — Return from Exception (Privileged Instruction)

Operation: If supervisor state
then (SP) → SR; SP + 2 → SP; (SP) → PC; SP + 4 → SP;
restore state and deallocate
stack according to (SP)
else TRAP

Assembler Syntax: RTE

Attributes: Unsized

Description: The processor state information in the exception stack frame on top of the stack is loaded into the processor. The stack format field in the format/offset word is examined to determine how much information must be restored.

Condition Codes: Set according to the content of the word on the stack.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	1	1

Format/Offset Word (in stack frame):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Format				0	0	Vector Offset									

Instruction Fields:

Format field — This 4-bit field defines the amount of information to be restored.

0000—Short Format, only four words are to be removed from the top of the stack. The status register and program counter are loaded from the stack frame.

0001—Throwaway Format, four words are removed from the top of stack. Only th status register is loaded, after which, the processor begins executing the RTE from the top of the active system stack. This format is used to mark the bottom of the interrupt stack.

0010—Instruction Error Format, six words are removed from the top of the stack. The first four words are used as in the Short Format and the remaining two words are thrown away.

1000—MC68010 Long Format, the MC68020 takes a format error exception.

1001—Coprocessor Mid-Instruction Format, 10 words are removed from the top of stack. Resumes coprocessor instruction execution.

1010—MC68020 Short Format, 16 words are removed from the top of the stack. Resumes instruction execution.

1011—MC68020 Long Format, 46 words are removed from the top of the stack. Resumes instruction execution.

Any others — the processor takes a format error exception.

RTM — Return from Module — RTM

Operation: Reload Saved Module State from Stack

Assembler Syntax: RTM Rn

Attributes: Unsized

Description: A previously saved module state is reloaded from the top of stack. After the module state is retrieved from the top of the stack, the caller's stack pointer is incremented by the argument count value in the module state.

Condition Codes: Set according to the content of the word on the stack.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	1	0	1	1	0	0	D/A	Register		

Instruction Fields:

D/A field — Specifies whether the module data pointer is in a data or an address register.
- 0—the register is a data register.
- 1—the register is an address register.

Register field — Specifies the register number for the module data area pointer which is to be restored from the saved module state. If the register specified is A7 (SP), the updated value of the register reflects the stack pointer operations, and the saved module data area pointer is lost.

RTR — Return and Restore Condition Codes — RTR

Operation: (SP) → CCR; SP + 2 → SP;
(SP) → PC; SP + 4 → SP

Assembler Syntax: RTR

Attributes: Unsized

Description: The condition codes and program counter are pulled from the stack. The previous condition codes and program counter are lost. The supervisor portion of the status register is unaffected.

Condition Codes: Set according to the content of the word on the stack.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	1	1

RTS — Return from Subroutine — RTS

Operation: (SP) → PC; SP + 4 → SP

Assembler Syntax: RTS

Attributes: Unsized

Description: The program counter is pulled from the stack. The previous program counter is lost.

Condition Codes: Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	0	1

SBCD — Subtract Decimal with Extend — SBCD

Operation: $Destination_{10} - Source_{10} - X \rightarrow Destination$

Assembler Syntax:
SBCD Dx,Dy
SBCD −(Ax), −(Ay)

Attributes: Size = (Byte)

Description: Subtract the source operand from the destination operand with the extend bit and store the result in the destination location. The subtraction is performed using decimal arithmetic. The operands may be addressed in two different ways:
1. Data register to data register: The operands are contained in the data registers specified in the instruction.
2. Memory tc memory: The operands are addressed with the predecrement addressing mode using the address registers specified in the instruction.

This operation is a byte operation only.

Condition Codes:

X	N	Z	V	C
*	U	*	U	*

- N Undefined.
- Z Cleared if the result is non-zero. Unchanged otherwise.
- V Undefined.
- C Set if a borrow (decimal) is generated. Cleared otherwise.
- X Set the same as the carry bit.

NOTE

Normally the Z condition code bit is set via programming before the start of an operation. This allows successful tests for zero results upon completion of multiple-precision operations.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	Register Dy/Ay			1	0	0	0	0	R/M	Register Dx/Ax		

Instruction Fields:

Register Dy/Ay field — Specifies the destination register.
If R/M = 0, specifies a data register.
If R/M = 1, specifies an address register for the predecrement addressing mode.

R/M field — Specifies the operand addressing mode:
0—The operation is data register to data register.
1—The operation is memory to memory.

Register Dx/Ax field — Specifies the source register:
If R/M = 0, specifies a data register.
If R/M = 1, specifies an address register for the predecrement addressing mode.

Scc — Set According to Condition — Scc

Operation: If Condition True
then 1s → Destination
else 0s → Destination

Assembler Syntax: Scc <ea>

Attributes: Size = (Byte)

Description: The specified condition code is tested; if the condition is true, the byte specified by the effective address is set to TRUE (all ones), otherwise that byte is set to FALSE (all zeroes). "cc" may specify the following conditions:

CC	carry clear	0100	$\bar{C}$
CS	carry set	0101	C
EQ	equal	0111	Z
F	never true	0001	0
GE	greater or equal	1100	$N \cdot V + \bar{N} \cdot \bar{V}$
GT	greater than	1110	$N \cdot V \cdot \bar{Z} + \bar{N} \cdot \bar{V} \cdot \bar{Z}$
HI	high	0010	$\bar{C} \cdot \bar{Z}$
LE	less or equal	1111	$Z + N \cdot \bar{V} + \bar{N} \cdot V$

LS	low or same	0011	C + Z
LT	less than	1101	$N \cdot \bar{V} + \bar{N} \cdot V$
MI	minus	1011	N
NE	not equal	0110	$\bar{Z}$
PL	plus	1010	$\bar{N}$
T	always true	0000	1
VC	overflow clear	1000	$\bar{V}$
VS	overflow set	1001	V

Condition Codes: Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	Condition				1	1	Effective Address					
										Mode			Register		

Instruction Fields:
Condition field — One of sixteen conditions discussed in description.
Effective Address field — Specifies the location in which the true/false byte is to be stored. Only data alterable addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

Note: 1. An arithmetic one and zero result may be generated by following the Scc instruction with a NEG instruction.

STOP — Load Status Register and Stop (Privileged Instruction) — STOP

Operation: If supervisor state
then Immediate Data → SR; STOP
else TRAP

Assembler Syntax: STOP #<data>

Attributes: Unsized

Description: The immediate operand is moved into the entire status register; the program counter is advanced to point to the next instruction and the processor stops fetching and executing instructions. Execution of instructions resumes when a trace, interrupt, or reset exception occurs. A trace exception will occur if the trace state is on when the STOP instruction begins execution. If an interrupt request is asserted with a priority higher than the priority level set by the immediate data, an interrupt exception occurs, otherwise, the interrupt request has no effect. If the bit of the immediate data corresponding to the S-bit is off, execution of the instruction will cause a privilege violation. External reset will always initiate reset exception processing.

Condition Codes: Set according to the immediate operand.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	1	0
Immediate Data															

Instruction Fields:
Immediate field — Specifies the data to be loaded into the status register.

SUB — Subtract Binary — SUB

Operation: Destination − Source → Destination

Assembler Syntax: SUB <ea>,Dn
SUB Dn,<ea>

Attributes: Size = (Byte, Word, Long)

Description: Subtract the source operand from the destination operand and store the result in the destination. The size of the operation may be specified to be byte, word, or long. The mode of the instruction indicates which operand is the source and which is the destination as well as the operand size.

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

N Set if the result is negative. Cleared otherwise.
Z Set if the result is zero. Cleared otherwise.
V Set if an overflow is generated. Cleared otherwise.
C Set if a borrow is generated. Cleared otherwise.
X Set the same as the carry bit.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	Register			Op-Mode			Effective Address Mode			Effective Address Register		

Instruction Fields:

Register field — Specifies any of the eight data registers.

Op-Mode field —

Byte	Word	Long	Operation
000	001	010	<Dn> − <ea> → <Dn>
100	101	110	<ea> − <Dn> → <ea>

Effective Address field — Determines addressing mode:

If the location specified is a source operand, then all addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An*	001	reg. number:An
(An)	010	reg. number:An
(An)+	011	reg. number:An
−(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

*For byte size operation, address register direct is not allowed.

If the location specified is a destination operand, then only alterable memory addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
−(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

Notes:
1. If the destination is a data register, then it cannot be specified by using the destination <ea> mode, but must use the destination Dn mode instead.
2. SUBA is used when the destination is an address register. SUBI and SUBQ are used when the source is immediate data. Most assemblers automatically make this distinction.

SUBA — Subtract Address — SUBA

Operation: Destination − Source → Destination

Assembler Syntax: SUBA <ea>,An

Attributes: Size = (Word, Long)

Description: Subtract the source operand from the destination address register and store the result in the address register. The size of the operation may be specified to be word or long. Word size source operands are sign extended to 32 bit quantities before the operation is done.

Condition Codes: Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	Register			Op-Mode			Effective Address Mode			Effective Address Register		

Instruction Fields:

Register field — Specifies any of the eight address registers. This is always the destination.

Op-Mode field — Specifies the size of the operation:
- 011—Word operation. The source operand is sign-extended to a long operand and the operation is performed on the address register using all 32 bits.
- 111—Long operations.

Effective Address field — Specifies the source operand. All addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	001	reg. number:An
(An)	010	reg. number:An
(An) +	011	reg. number:An
−(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

SUBI — Subtract Immediate — SUBI

Operation: Destination − Immediate Data → Destination

Assembler Syntax: SUBI #<data>,<ea>

Attributes: Size = (Byte, Word, Long)

Description: Subtract the immediate data from the destination operand and store the result in the destination location. The size of the operation may be specified to be byte, word, or long. The size of the immediate data matches the operation size.

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

- N Set if the result is negative. Cleared otherwise.
- Z Set if the result is zero. Cleared otherwise.
- V Set if an overflow is generated. Cleared otherwise.
- C Set if a borrow is generated. Cleared otherwise.
- X Set the same as the carry bit.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	0	Size		Effective Address Mode			Effective Address Register		
Word Data								Byte Data							
Long Data															

Instruction Fields:

Size field — Specifies the size of the operation.
- 00—byte operation.
- 01—word operation.
- 10—long operation.

Effective Address field — Specifies the destination operand. Only data alterable addressing modes are allowed as shown:

SUBI — Subtract Immediate — SUBI

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
−(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

Immediate field — (Data immediately following the instruction)
If size = 00, then the data is the low order byte of the immediate word.
If size = 01, then the data is the entire immediate word.
If size = 10, then the data is the next two immediate words.

SUBQ — Subtract Quick — SUBQ

Operation: Destination − Immediate Data → Destination

Assembler Syntax: SUBQ #<data>,<ea>

Attributes: Size = (Byte, Word, Long)

Description: Subtract the immediate data from the destination operand. The data range is from 1-8. The size of the operation may be specified to be byte, word, or long. Word and long operations are also allowed on the address registers and the condition codes are not affected. When subtracting from address registers, the entire destination address register is used, regardless of the operation size.

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

N Set if the result is negative. Cleared otherwise.
Z Set if the result is zero. Cleared otherwise.
V Set if an overflow is generated. Cleared otherwise.
C Set if a borrow is generated. Cleared otherwise.
X Set the same as the carry bit.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	Data			1	Size		Effective Address					
										Mode			Register		

Instruction Fields:
Data field — Three bits of immediate data, 0, 1-7 representing a range of 8, 1 to 7 respectively.
Size field — Specifies the size of the operation:
00—byte operation.
01—word operation.
10—long operation.
Effective Address field — Specifies the destination location. Only alterable addressing modes are allowed as shown:

SUBQ — Subtract Quick — SUBQ

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An*	001	reg. number:An
(An)	010	reg. number:An
(An)+	011	reg. number:An
−(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

*Word and long only.

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

SUBX — Subtract with Extend — SUBX

Operation: Destination − Source − X → Destination

Assembler Syntax:
SUBX Dx,Dy
SUBX −(Ax), −(Ay)

Attributes: Size = (Byte, Word, Long)

Description: Subtract the source operand from the destination operand along with the extend bit and store the result in the destination location. The operands may be addressed in two different ways:

1. Data register to data register: The operands are contained in data registers specified in the instruction.
2. Memory to memory. The operands are contained in memory and addressed with the predecrement addressing mode using the address registers specified in the instruction.

The size of the operand may be specified to be byte, word, or long.

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

- N Set if the result is negative. Cleared otherwise.
- Z Cleared if the result is non-zero. Unchanged otherwise.
- V Set if an overflow is generated. Cleared otherwise.
- C Set if a carry is generated. Cleared otherwise.
- X Set the same as the carry bit.

NOTE

Normally the Z condition code bit is set via programming before the start of an operation. This allows successful tests for zero results upon completion of multiple-precision operations.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	Register Dy/Ay			1	Size		0	0	R/M	Register Dx/Ax		

Instruction Fields:

Register Dy/Ay field — Specifies the destination register:
If R/M = 0, specifies a data register.
If R/M = 1, specifies an address register for the predecrement addressing mode.

Size field — Specifies the size of the operation:
00—byte operation.
01—word operation.
10—long operation.

SUBX — Subtract with Extend — SUBX

R/M field — Specifies the operand addressing mode:
  0—The operation is data register to data register.
  1—The operation is memory to memory.
Register Dx/Ax field — Specifies the source register:
  If R/M = 0, specifies a data register.
  If R/M = 1, specifies an address register for the predecrement addressing mode.

SWAP — Swap Register Halves — SWAP

Operation: Register [31:16] ↔ Register [15:0]

Assembler Syntax: SWAP Dn

Attributes: Size = (Word)

Description: Exchange the 16-bit halves of a data register.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

N Set if the most significant bit of the 32-bit result is set. Cleared otherwise.
Z Set if the 32-bit result is zero. Cleared otherwise.
V Always cleared.
C Always cleared.
X Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	1	0	0	0	Register		

Instruction Fields:
Register field — Specifies the data register to swap.

TAS — Test and Set an Operand — TAS

Operation: Destination Tested → Condition Codes; 1 → bit 7 of Destination

Assembler Syntax: TAS <ea>

Attributes: Size = (Byte)

Description: Test and set the byte operand addressed by the effective address field. The current value of the operand is tested and N and Z are set accordingly. The high order bit of the operand is set. The operation is indivisible (using a read-modify-write memory cycle) to allow synchronization of several processors.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

N Set if the most significant bit of the operand was set. Cleared otherwise.
Z Set if the operand was zero. Cleared otherwise.
V Always cleared.
C Always cleared.
X Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	0	1	1	Effective Address Mode			Effective Address Register		

Instruction Fields:
Effective Address field — Specifies the location of the tested operand. Only data alterable addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
−(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—
(bd,PC,Xn)	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

TRAP — Trap — TRAP

Operation: SSP − 2 → SSP; Format/Offset → (SSP);
SSP − 4 → SSP; PC → (SSP); SSP − 2 → SSP;
SR → (SSP); Vector Address → PC

Assembler Syntax: TRAP #<vector>

Attributes: Unsized

Description: The processor initiates exception processing. The vector number is generated to reference the TRAP instruction exception vector specified by the low order four bits of the instruction. Sixteen TRAP instruction vectors (0-15) are available.

Condition Codes: Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	0	Vector			

Instruction Fields:
Vector field — Specifies which trap vector contains the new program counter to be loaded.

TRAPcc — Trap on Condition — TRAPcc

Operation: If cc then TRAP

Assembler Syntax: TRAPcc
TRAPcc.W #<data>
TRAPcc.L #<data>

Attributes: Unsized or Size = (Word, Long)

Description: If the selected condition is true, the processor initiates exception processing. The vector number is generated to reference the TRAPcc exception vector. The stacked program counter points to the next instruction. If the selected condition is not true, no operation is performed, and execution continues with the next instruction in sequence. The immediate data operand(s) is placed in the next word(s) following the operation word and is (are) available for user definition for use within the trap handler. "cc" may specify the following conditions.

CC	carry clear	0100	$\overline{C}$	LS	low or same	0011	C+Z
CS	carry set	0101	C	LT	less than	1101	$N\cdot\overline{V}+\overline{N}\cdot V$
EQ	equal	0111	Z	MI	minus	1011	N
F	never true	0001	0	NE	not equal	0110	$\overline{Z}$
GE	greater or equal	1100	$N\cdot V+\overline{N}\cdot\overline{V}$	PL	plus	1010	$\overline{N}$
GT	greater than	1110	$N\cdot V\cdot\overline{Z}+\overline{N}\cdot\overline{V}\cdot\overline{Z}$	T	always true	0000	1
HI	high	0010	$\overline{C}\cdot\overline{Z}$	VC	overflow clear	1000	$\overline{V}$
LE	less or equal	1111	$Z+N\cdot\overline{V}+\overline{N}\cdot V$	VS	overflow set	1001	V

Condition Codes: Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	Condition				1	1	1	1	1	Op-Mode		
Optional Word															
or Long Word															

Instruction Fields:
Condition field — One of sixteen conditions discussed previously.
Op-Mode field — Selects the instruction form.
010—Instruction is followed by one operand word.
011—Instruction is followed by two operands words.
100—Instruction has no following operand words.

TRAPV — Trap on Overflow — TRAPV

Operation: If V then TRAP

Assembler Syntax: TRAPV

Attributes: Unsized

Description: If the overflow condition is set, the processor initiates exception processing. The vector number is generated to reference the TRAPV exception vector. If the overflow condition is clear, no operation is performed and execution continues with the next instruction in sequence.

Condition Codes: Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	1	0

TST — Test an Operand — TST

Operation: Destination Tested → Condition Codes

Assembler Syntax: TST <ea>

Attributes: Size = (Byte, Word, Long)

Description: Compare the operand with zero. No results are saved; however, the condition codes are set according to results of the test. The size of the operation may be specified to be byte, word, or long.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

- N Set if the operand is negative. Cleared otherwise.
- Z Set if the operand is zero. Cleared otherwise.
- V Always cleared.
- C Always cleared.
- X Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	0	Size		Effective Address Mode			Register		

Instruction Fields:

Size field — Specifies the size of the operation:
- 00—byte operation.
- 01—word operation.
- 10—long operation.

Effective Address field — Specifies the destination operand. If the operation size is word or long, all addressing modes are allowed. If the operation size is byte, only data addressing modes are allowed as shown:

Addr. Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An)+	011	reg. number:An
−(An)	100	reg. number:An
(d_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An
(bd,An,Xn)	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

Addr. Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	111	010
(d_8,PC,Xn)	111	011
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

UNLK — Unlink — UNLK

Operation: An → SP; (SP) → An; SP + 4 → SP

Assembler Syntax: UNLK An

Attributes: Unsized

Description: The stack pointer is loaded from the specified address register. The address register is then loaded with the long word pulled from the top of the stack.

Condition Codes: Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	1	1	Register		

Instruction Fields:

Register field — Specifies the address register through which the unlinking is to be done.

UNPK — Unpack BCD — UNPK

Operation: Source (Packed BCD) + adjustment → Destination (Unpacked BCD)

Assembler Syntax: UNPACK –(Ax), –(Ay),#<adjustment>
UNPK Dx,Dy,#<adjustment>

Attributes: Unsized

Description: In the unpack operation, two BCD digits within the byte source operand are separated into two bytes with the BCD digit residing in the lower nibble and 0 in the upper nibble. The adjustment is then added to this unpacked value without affecting the condition codes.

When both operands are data registers, the source register contents are unpacked, the extension word is added, and the result is placed in the destination register. The high word of the destination register is unaffected.

Source:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Dx	u	u	u	u	u	u	u	u	a	b	c	d	e	f	g	h

Intermediate Expansion:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	a	b	c	d	0	0	0	0	e	f	g	h

Add Adjustment Word:

15–0
16-Bit Extension

Destination:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Dy	v	v	v	v	a'	b'	c'	d'	w	w	w	w	e'	f'	g'	h'

When the addressing mode specified is predecrement, two BCD digits are extracted from a byte at the source address. After adding the extension word, two bytes are then written to the destination address.

Source:

	7	6	5	4	3	2	1	0
	a	b	c	d	e	f	g	h
(Ax)								

— Continued —

UNPK — Unpack BCD — UNPK

Intermediate Expansion:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	a	b	c	d	0	0	0	0	e	f	g	h

Add Adjustment Word:

15–0
16-Bit Extension

Destination:

	7	6	5	4	3	2	1	0
	v	v	v	v	a'	b'	c'	d'
	w	w	w	w	e'	f'	g'	h'
(Ay)								

Condition Codes: Not affected.

Instruction Format:

15	14	13	12	11–9	8	7	6	5	4	3	2–0
1	0	0	0	Register Dy/Ay	1	1	0	0	0	R/M	Register Dx/Ax
Adjustment											

Instruction Fields:

Register Dy/Ay field — Specifies the destination register.
  If R/M = 0, specifies a data register.
  If R/M = 1, specifies an address register for the predecrement addressing mode.
R/M field — Specifies the operand addressing mode.
  0—The operation is data register to data register.
  1—The operation is memory to memory.
Register Dx/Ax field — Specifies the data register.
  If R/M = 0, specifies a data register.
  If R/M = 1, specifies an address register for the predecrement addressing mode.
Adjustment field — Immediate data word which is added to the source operand.
  Appropriate constants can be used to translate from BCD to ASCII or EBCDIC.

APPENDICE E

RIFERIMENTI BIBLIOGRAFICI

CAPITOLO 1

I prodotti descritti in questo capitolo non rappresentano che un modesto campionario di tutti quelli che impiegano il processore MC68020 della Motorola. I nuovi prodotti basati sull'MC68020 sono annunciati mensilmente in molte riviste di elettronica e di computer, quali *Byte*, *Mini-Micro Systems*, *IEEE Computer*, e *IEEE Micro*. (Le pubblicazioni dell'IEEE sono disponibili presso l'Institute of Electrical and Electronic Engineers, Inc.)

I prodotti della famiglia dell'MC68020 sono descritti in un certo numero di pubblicazioni ed in altri "ausilii" all'addestramento offerti dalla Motorola. I corsi audio elencati con i riferimenti consistono ciascuno di tre cassette e sono rivolti a coloro che hanno una certa familiarità con l'MC68000. Gli annunci di nuovi prodotti compaiono sulla rivista *Motorola Semiconductor Data Update*. Il catalogo *16/32-Bit Microcomputer System Components* elenca tutti i prodotti della Motorola che riguardano la famiglia dell'MC68020. *The Source* è un catalogo di prodotti software per la famiglia dell'MC68020, inclusi i sistemi operativi, il software di sviluppo, i programmi applicativi offerti da fornitori indipendenti oltre che dalla Motorola. L'articolo di Johnson descrive i membri della famiglia dell'MC68000 a 16 bit e della famiglia dell'MC68020 a 32 bit. Le pubblicazioni sono disponibili presso la Motorola, Inc., di Phoenix in Arizona, U.S.A.

Gli altri articoli discutono vari prodotti o applicazioni che si basano sul processore MC68020. L'articolo di Turnick, ad esempio, discute il sistema di sviluppo HP64000-UX della Hewlett-Packard. Esso è utilizzato per sviluppare e collaudare l'hardware ed il software di prodotti che impiegheranno l'MC68020.

Motorola

Corsi su audiocassette, *An Introduction to the MC68020*, Motorola Publication MTTA2, e *An Introduction to the MC68030*, (MTTA3). Phoenix, Arizona: Motorola, Inc.

Johnson, Thomas L., "A comparison of MC68000 Family Processors", *Byte*, 11, n. 9 (settembre 1986), 205-218

Motorola Semiconductor Data Update, Phoenix, Arizona: Motorola, Inc.

16/32-Bit Microcomputer System Components. Phoenix, Arizona: Motorola, Inc.

The Source, Phoenix, Arizona: Motorola, Inc.

Generali

Allison, Andrew, "Multiprocessors Boost System Power", *Mini- Micro Systems*, XX, n. 5 (maggio 1987), 105-115.

Mokhoff, Nicolas, "Supermicro Look-Alikes Differ below the Surface in Processing Power", *Computer Design*, 25, n. 16 (1 settembre 1986), 57-75.

Ohr, Stephan, "CAE", *Electronic Design*, 34, n. 23 (2 ottobre 1986), 67-74.

Tunick, Diane, "32-Bit Tools Target 68020 Development", *Electronic Design*, 34, n. 28 (27 novembre 1986), 41-42.

Williams, Gregg e Tom Thompson, "The Apple Macintosh II", *Byte*, 12, n. 4 (aprile 1987), 85-106.

CAPITOLO 2

Un certo numero di articoli riguardanti la famiglia dell'MC68020 sono stati pubblicati nelle rivesite di computer. Il lettore dovrebbe consultare gli ultimi numeri di tali riviste per tenersi aggiornato sui rapidi sviluppi della tecnologia dei microprocessori. I due articoli elencati di seguito sono un'introduzione all'MC68020 ed alle sue capacità per svariate applicazioni. I riferimenti bibliografici contenenti dettagli specifici sulle caratteristiche dell'MC68020 sono citati nei prossimi capitoli, relativamente agli argomenti presentati nel testo.

Beims, Bob, *The MC68020 32-Bit MPU: Opening New Application Doors*, Motorola Publication AR232. Phoenix, Arizona: Motorola, Inc.

MacGregor, Doug, Dave Mothersole e Bill Moyer, "The Motorola MC68020", *IEEE Micro*, 4, n. 4 (agosto 1984), 101-108.

CAPITOLO 3

I libri di Stein e Munro, Hwang e Sterbenz elencati qui presentano un punto di vista rigorosamente matematico di rappresentazioni numeriche. Sterbenz si concentra sulla notazione della virgola mobile. Gli articoli di *IEEE Computer* sono consigliati per informazioni sul formato standard dell'IEEE.

Weller presenta un certo numero di tecniche di conversione tra tipi di dati, che possono essere utili per il programmatore in linguaggio assembler. Il progetto dell'hardware del computer e dell'impiego di rappresentazioni numeriche per le operazioni aritmetiche è discusso da AbdAlla e da Meltzer, mentre Mackenzie discute i dettagli di un gran numero di codici per la rappresentazione di caratteri.

Abd-Alla, Abd-Elfattah M. e Arnold C. Melzter, *Principles of Digital Computer Design*, vol. I, Englewood Cliffs, New Jersey: Prentice-Hall, 1976. (Il cap. 4 presenta un certo numero di codici.)

Coonen, Jerome T., "An Implementation Guide to a Proposal Standard for Floating Point Arithmetic", *IEEE Computer*, 13, n. 1 (gennaio 1980), 68-79.

Hwang, Kai, *Computer Arithmetic*, New York: Wiley, 1979.

IEEE Computer, 14, n. 3 (marzo 1981). (Vari articoli in questo numero della rivista discutono lo standard della virgola mobile.) Lo standard stesso è ANSI/IEEE Std 754-1985.

Mackenzie, Charles E., *Coded Character Sets: History and Development*, Reading, Massachussets: Addison-Wesley, 1980.

Stein, Marvin L., e William D. Munro, *Introduction to Machine Arithmetic*, Reading, Massachussets: Addison-Wesley, 1971.

Sterbenz, Pat H., *Floating-Point Computation*, Englewood Cliffs, New Jersey: Prentice-Hall, 1974.

Weller, Walter J., *Assembly Language Programming for Small Computers*, Lexington, Massachussets, Lexington Books, 1975.

CAPITOLO 4

Gli articoli ed i libri di testo elencati qui trattano vari aspetti del progetto dei moderni sistemi di computer e dei chip di CPU. Gli articoli di Farrell, Johnson, Kuban,

ed il gruppo di semiconduttori della Motorola trattano specificamente la famiglia di prodotti della Motorola. Guterl e Silbey ed altri confrontano i vari metodi di progettazione di una CPU. Il libro di testo di Tredennick spiega il progetto di microprocessori impiegano il micro/370 dell'IBM e l'MC68000 come esempi. Egli è stato uno dei progettisti della CPU MC68000. Il libro di testo di Stone fornisce un'analisi dettagliata di molte caratteristiche dei moderni microprocessori. Il libro tratta in un certo dettaglio i pipeline e la memoria cache.

Farrell, James, III, "The Advancing Technology of Motorola's Microprocessors and Microcomputers", *IEEE Micro*, 4, n. 5 (ottobre 1984), 55-63.

Guterl, Fred, "Chip Architecture: A Revolution Brewing", *IEEE Spectrum*, 20, n. 7 (luglio 1983), 30-37.

Johnson, Thomas L. "The RISC/CISC Melting Pot", *Byte*, 12, n. 4 (aprile 1987), 153-160.

Kuban, John R., e John E. Salick, *Testing Approaches in the MC68020, Motorola Publication AR255*, Austin, Texas: Motorola Semiconductor Products, Inc.

MC68000 Microprogrammed Architecture, Motorola Publication AR235, Austin, Texas: Motorola Semiconductor Products, Inc.

MC68020 Technical Summary, Motorola Publication BR243, Austin, Texas: Motorola Semiconductor Products, Inc.

Silbey, A., T. Milutinovic e V. Mendoza-Grado, "A Survey of Advanced Microprocessors and HLL Computer Architectures", *IEEE Computer*, 19, n. 8 (agosto 1986), 72-85

Stone, Harold S., *High-Performance Computer Architecture*, Reading, Massachussets, Addison-Wesley, 1987.

Tredennick, Nick, *Microprocessor Logic Design*, Bedford, Massachussets, Digital Press, 1987.

CAPITOLO 5

Il breve ma brillante testo di Barron descrive in dettaglio il funzionamento di un'assemblatore. Molti dei libri di testo scritti per il computer VAX della Digital Equipment Company sono utili per comprendere la capacità dell'MC68020 e della sua programmazione in linguaggio assembler. I testi di Levi ed Eckhouse e di Peters non sono che due esempi. I due assemblatori utilizzati dall'autore sono descritti nei manuali della Motorola e della Quelo.

Barron, D. W., *Assemblers and Loaders*, seconda ediz., New York: American Elsevier, 1972.

Levy, Henry M. e Richard H. Echkhouse, Jr., *Computer Programming and Architecture: The VAX-11*, Bedford, Massachussets: Digital Press, 1980.

MC68000 Family Resident Structured Assembler, Tempe, Arizona, Motorola, Inc. Microsystems.

Peters, J. F., III, *The Art of Assembly Language Programming: VAX-11*, Reston, Virginia: Reston, 1985.

Quelo 68000 Software Development Tools, Seattle, Washington: Quelo, Inc.

CAPITOLO 6

In un certo numero di opere è discussa la programmazione in linguaggio assembler, in una maniera utile per il programmatore dell'MC68020. In particolare, il libro di Kane ed altri autori, elencato di seguito, fornisce un certo numero di buoni esempi di programmi per l'MC68000. Tali programmi sono eseguibili su un computer basato sull'MC68020. Molti libri di testo che riguardano il sistema PDP-11 possono servire nello studio dell'MC68020 se si tiene conto delle differenze tra i processori. Per esempio, l'istruzione

```
CMP        A,B
```

per il PDP-11 valuta (A)–(B), cioè l'opposto di quanto avviene nell'MC68020. I testi di Echkouse e Morris e di Lewis spiegano molti aspetti della programmazione in linguaggio assembler per il PDP-11. I programmi per il PDP-11 possono essere convertiti per l'MC68020, e viceversa, da un programmatore che sia a conoscenza di entrambe le macchine. Nei riferimenti bibliografici relativi al cap. 5 sono elencate altre pubblicazioni che forniscono ulteriori dettagli sulla programmazione in linguaggio assembler e sul funzionamento degli assemblatori.

Eckhouse, Richard H., Jr, e L. R. Morris, *Minicomputer Systems*, seconda ediz., Englewood Cliffs, New Jersey, Prentice-Hall, 1979.

Kane, Gerald, Doug Hawkins, e Lance Leventhal, *68000 Assembly Language Programming*, Berkeley, California: Osborne (McGraw- Hill), 1981.

Lewis, Harry R., *An Introduction to Computer Programming and Data Structures Using MACRO-11*, Reston, Virginia: Reston, 1981.

CAPITOLO 7

Il volume di Knuth sugli algoritmi seminumerici contiene un certo numero di algoritmi utili ed altre informazioni per coloro che si occupano di programmazione matematica avanzata. Il libro di testo di Stein e Monro presenta una base rigorosa per l'aritmetica di macchina.

Knuth, Donald E., *The Art of Computer Programming*, vol. II: *Seminumerical Algorithms*, Reading, Massachussets: Addison- Wesley, 1968.

Stein, Marvin L., e William D. Monro, *Introduction to Machine Arithmetic*, Reading, Massachussets: Addison-Wesley, 1971.

CAPITOLO 8

Un certo numero di libri di testo elencati nei riferimenti relativi ai capitoli precedenti tratta le operazioni logiche e sui bit (per esempio, le pubblicazioni di Eckhouse e Morris e di Wakerly). La matematica dello scorrimento è discussa in Stein e Munro, nella bibliografia per il cap. 3.

L'articolo di Biems elencato qui discute alcuni esempi dell'impiego delle istruzioni di campo di bit. Deitel descrive in dettaglio la mappa di bit del sistema operativo CP/M. Baase fornisce degli esempi dell'uso delle istruzioni di campo di bit per il computer VAX, incluso un programma di grafica con mappa di bit.

Baase, Sara, *VAX-11 Assembly Language Programming*, Englewood Cliffs, New Jersey, Prentice-Hall, 1983.

Biems, Bob, *The MC68020 and System V/68*, Motorola Publication AR219. Phoenix, Arizona: Motorola, Inc.

Deitel, Harvey, M., *An Introduction to Operating Systems*, Reading, Massachussets: Addison-Wesley, 1984.

CAPITOLO 9

La codifica indipendente dalla posizione è discussa in molti libri di testo riguardanti il computer PDP-11. In particolare, Tanenbaum spiega questa tecnica ed include anche delle discussioni sull'indirizzamento dei registri di base e sulla rilocazioni dinamica dei programmi. Stritter e Gunter discutono un certo numero di istruzioni impiegate in questo capitolo. Knuth fornisce un'eccellente discussione delle strutture di dati, come pure dell'impiego di subroutine con enfasi sulla programmazione in linguaggio assembler. La tecnica del frame di stack è discussa a fondo nel libro di Wakerly, che comprende alcuni esempi di programmazione dell'MC68000.

Knuth, Donald E., *The Art of Computer Programming*, vol. I: *Fundamental Algorithms*, Reading, Massachussets: Addison-Wesley, 1969.

Stritter, Edward, e Tom Gunter, "A Microprocessor Architecture for a Changing World: The Motorola 68000", *IEEE Computer*, 12, n. 2 (febbraio 1979), 43-52.

Tanenbaum, Andrew S., *Structured Computer Organization*, seconda ediz., Englewood Cliffs, New Jersey, Prentice-Hall, 1984.

Wakerly, John F., *Microcomputer Architecture and Programming.* New York: Wiley, 1981.

CAPITOLO 10

Per un sistema di computer specifico, si dovrebbero consultare i manuali per l'utente allo scopo di ottenere informazioni dettagliate del funzionamento del sistema, al livello discusso in questo capitolo. La porzione di un sistema operativo che controlla direttamente la CPU è talvolta denominata *kernel* (nucleo) del sistema operativo. Una descrizione del kernel del computer VAX-11 della Digital Equiment Corporation è fornita nel libro di testo di Levy ed Ackhouse elencato di seguito.

Levy, Henry M. e Richard H. Echkhouse, Jr., *Computer Programming and Architecture: The VAX-11*, Bedford, Massachussets: Digital Press, 1980.

CAPITOLO 11

Le eccezioni riconosciute dai processori a 32 bit della Motorola sono descritte in dettaglio nei manuali per l'utente di quei processori. Le eccezioni che sono causate o riconosciute dai coprocessori MC68851, MC68881 o MC68882 sono trattate nei rispettivi manuali per l'utente. Le eccezioni definite specificamente per l'unità di gestione della memoria sul chip dell'MC68030 sono illustrate nel libro *MC68030 User's Manual.* Le eccezioni per la CPU MC68030 sono discusse anche nel cap. 15 di questo libro di testo.

CAPITOLO 12

Di seguito è elencato un certo numero di lavori sugli argomenti introdotti in questo capitolo. Per favorire il lettore, i riferimenti bibliografici sono stati suddivisi in categorie: generali, di coprocessori in virgola mobile, di gestione della memoria, e di multielaborazione. Per ulterori dettagli concernenti i chip MC68020, MC68851, MC68881 e MC68882, si dovrebbe consultare lo *User's Manual* del particolare chip. Questi manuali sono disponibili dalla Motorola o dalla Prentice-Hall.

Due articoli di Ripps e Mushinsky trattano le caratteristiche generali dell'MC68020, considerandone anche le prestazioni. L'articolo di Van Loo discute la progettazione di sistemi di memoria cache nei sistemi basati dull'MC68020.

Il libro di testo di Hamming è un'introduzione all'argomento dei metodi numerici. In questo libro sono descritti molti algoritmi utili che si possono programmare

per il coprocessore in virgola mobile MC68881. L'articolo di Harris e Johnson descrive l'impiego di macroistruzioni per emulare l'MC68881. Il coprocessore è descritto in dettaglio nell'articolo di Huntsman e Cawthron. Una definizione precisa degli standard dell'IEEE per l'aritmetica in virgola mobile è reperibile in Std 754-1985.

I dettagli riguardanti le tecniche di gestione della memoria, inclusi i sistemi di memoria virtuale, sono descritte negli articoli di Cohen e McGarity, di Cruess, e di Fuhrt e Milutinovic. L'ultimo articolo descrive un certo numero di metodi di gestione della memoria, incluso quello impiegato in sistemi basati sull'MC68020 con un'unità MC68851. Il libro di testo di Deitel svolge delle considerazioni di gestione della memoria in merito al progetto di sistemi operativi.

La capacità generale di multielaborazione dell'MC68020 è descritta nell'articolo di Beims. Il libro di testo di Stone discute molti dettagli inerenti la progettazioni di sistemi multiprocessore. Sono analizzate le istruzioni di test e assegnazione (TAS) e di confronto e scambio (CAS), evidenziandone in modo particolare le possibilità di un impiego errato. Il libro di Stone presenta anche una discussione approfondita della memoria cache e della memoria virtuale.

Generali

Ripps, David, e B. Mushinsky, "32-Bit <F128M>m<F255D>P Speeds Code Design and Execution", *EDN*, 30, n. 15 (27 giugno 1985), 163-168.

Ripps, David, e B. Mushinsky, "Benchmark Contrast 68020 Cache-Memory Operations", *EDN*, 30, n. 18 (8 agosto 1985), 177-202.

Van Loo, William, "Maximize Performance by Choosing Best Memory", *Computer Design*, 26, n. 14 (1 agosto 1987), 89-94.

Coprocessori in virgola mobile

Hamming, R. W., *Numerical Methods for Scientists and Engineers*, seconda ediz., New Yrk: Mc-Graw-Hill, 1962.

Harris, Sarah, e T. Johnson, *Software Links Mathchip to 68000-Family μPs*, Motorola Publication AR233. Tempe, Arizona: Motorola, Inc.

Huntsman, Clayton, e D. Cawthron, "The MC68881 Floating-Point Coprocessor", *IEEE Micro*, 3, n. 6 (dicembre 1983), 44-54.

IEEE Standard for Binary Floating-Point Arithmetic, Std 754-1985. New York: IEEE, 1985.

Gestione della memoria

Cohen, Brad, e R. McGarity, "The Design and Implementation of the MC68851 Paged Memory Management Unit", *IEEE Micro*, 6, n. 2, (aprile 1986), 13-28.

Cruess, Michael W. "Memory Management Chip for 68020 Translates Addresses in Less Than a Clock Cycle", *Electronic Design*, 34, n. 11 (15 maggio 1985), 151-162.

Deitel, Harvey M., *An Introduction to Operating Systems, Reading, Massachussets: Addison-Wesley, 1984.*

Fuhrt, Borivoje, e V. Milutinovic, "A Survey of Microprocessor Architectures for Memory Management", *IEEE Computer*, 20, n. 3, (marzo 1987), 48-67.

Multielaborazione

Beims, Bob, *Multiprocessing Capabilities of the MC68020 32-Bit Microprocessor*, Motorola Publication AR220. Tempe, Arizona: Motorola, Inc.

Stone, Harold S., *High-Performance Computer Architecture*, Reading, Massachussets: Addison-Wesley, 1987.

CAPITOLO 13

Una breve descrizione di ogni chip periferico disponibile per la famiglia di microprocessori dell'MC68020 è fornita nella *Motorola Semiconductor Master Selection Guide*. Per ulteriori dettagli in merito ad un particolare chip, si dovrebbero consultare lo *User's Manual* o le specifiche tecniche del particolare chip. Questi documenti sono disponibili dalla Motorola. L'MC68901 è descritto nell'articolo di Curran e Folkes. Il precedente libro di testo dell'autore (Harman e Lawson) descrive le tecniche di programmazione di I/O per vari chip periferici della famiglia dell'MC68000.

L'articolo di MacGregor e Rubinstein svolge alcune considerazioni di temporizzazione per l'esecuzione di un programma da parte dell'MC68020. L'articolo di Olsen discute il pipeline dell'MC68020 ed il suo effetto sulle prestazioni della CPU.

Il libro di testo di Slater considera molti aspetti della progettazione dei circuiti dell'hardware, includendo alcune caratteristiche dei chip della famiglia di 16 bit dell'MC68000. Una descrizione dell'arbitrato di bus impiegando il chip MC68453 è contenuta nella descrizione del prodotto (ADI-696) della Motorola. L'articolo di Hoffman e Tietjen descrive le caratteristiche d'interfacciamento dell'MC68881.

Curran, Tim, e Don Folkes, "68000 Peripheral Chips Assume I/O Tasks and More", *Electronic Design*, 31, n. 21 (13 ottobre 1983), 123-128.

Harman, Thomas L., e Barbara Lawson, *The Motorola MC68000 Microprocessor Family*, Englewood Cliffs, New Jersey, Prentice-Hall, 1985.

Hoffman, Bruce, e Donald Tietjen, "Floating Point Unit Extends Arithmetic Processing", *Digital Design*, 15, n. 12 (dicembre 1985), 61-64.

MacGregor, Doug, e Jon Rubinstein, "A Performance Analysis of MC68020-Based Systems", *IEEE Micro*, 5, n. 6, (dicembre 1985), 50-70.

MC68452-Bus Arbitration Module, Product Information ADI-696. Tempe, Arizona: Motorola, Inc., 1982.

Motorola Semiconductor Master Selection Guide. Tempe, Arizona: Motorola, Inc.

Olsen, David, "Effects of Pipelining on Algorithms for the MC68020", *Digital Design*, 15, n. 6, (giugno 1985), 72-73.

Slater, Michael, *Microprocessor-Based Design*. Mountain View, California: Mayfield Publishing Company, 1987.

CAPITOLO 14

Il VMEbus è descritto completamente nella *VMEbus Specification* elencata di seguito nei riferimenti. *VMEbus Systems* è una pubblicazione bimestrale rivolta a presentare articoli informativi ed altre informazioni riguardanti il VMEbus e i relativi prodotti. L'organizzazione VITA (*VMEbus International Trade Association*) assiste gli utenti interessati del VMEbus: i membri ricevono informazioni su tali prodotti. L'articolo di Wayne Fisher descrive il VMEbus piuttosto dettagliatamente.

Altri bus impiegati coi sistemi VMEbus sono descritti nelle specifiche e negli articoli elencati nei riferimenti per i bus VSB e VMS. Ulteriori informazioni in merito a questi bus sono disponibili dai rappresentanti della Motorola.

I prodotti del VMEbus della Motorola sono descritti nell'opuscolo BR606 della Motorola, Inc. Questa società ed altre case produttrici di VMEbus forniscono le informazioni più aggiornate sulle loro linee di prodotti in cataloghi e descrizioni disponibili presso i rispettivi rappresentanti.

I programmi di benchmark e i confronti delle prestazioni per i microprocessori sono presentati negli articoli di Cooper ed altri autori e di McCallum e Chua. L'MC68020 viene confrontato con altri processori di 32 bit in questi studi di benchmark. Ron Wilson discute i vari metodi di selezione e valutazione del sistema.

Altri bus e prodotti che non fanno parte della famiglia dell'MC68020 sono descritti in vari articoli elencati di seguito, alla voce "Altri bus e prodotti". L'articolo di Borrill confronta diversi bus noti di microcomputer. Una descrizione del prodotto di Falk spiega il software (Hunter Systems XDOS) per convertire i programmi di MS-DOS nel codice dell'MC68020. Il coprocessore Definicon ed il relativo software sono descritti nei due articoli di Marshall ed altri autori, mentre Dave Wilcox descrive l'impiego del suo coprocessore Definicon con un personal computer per risparmiare tempo di esecuzione su un supercomputer Cray! Il suo articolo è una dimostrazione di ciò che può ottenere un brillante progettista di sistemi quando conosce a fondo sia l'applicazione che il sistema del computer e riesce ad integrare tali conoscenze.

VMEbus

Fischer, Wayne, "IEEE P1014 - A Standard for the High_performance VMEbus", *IEEE Micro*, 5, n. 1 (febbraio 1985), 31-41.

VMEbus Specification, disponibile presso la Motorola, Inc., e da altre fonti.

VMEbus International Trade Association (VITA), Scottsdale, Arizona.

VMEbus Systems, pubblicazione bimestrale, Motorola, Inc., Tempe, Arizona.

Bus VSB e VMS

MacKenna, Craig, "VMS How This Serial Bus Works - amd How to Use It", *VMEbus Systems*, 3, n. 1 (gennaio-febbraio 1987), 33-40.

Pri-Tal, Shlomo, e Doug Kraft, "VSB: a Secondary Bus for VMEbus Systems", *VMEbus Systems*, 2, n. 3 (novembre-dicembre 1986), 22-33.

VME Subsystem BUS (VSB) Specification, Tempe, Arizona: Motorola, Inc.

Moduli e software del VMEbus

VMEbus Products, Motorola Publication BR606. Tempe, Arizona: Motorola, Inc.

Selezione e prestazioni del sistema

Cooper, Thayne C., Wayne D. Bell, Frank C. Lin, e Norm J. Rasmussen, "A Benchmark Comparison of 32-Bit Microprocessors", *IEEE Micro*, 6, n. 4 (agosto 1986), 53-58.

McCallum, John C., e Tat-Seng Chua, "A Synthetic Instruction Mix for Evaluating Microprocessor Performance", *IEEE Micro*, 7, n. 3 (giugno 1987), 63-80.

Wilson, Ron, "Choosing a Microprocessor: Designers Take Many Paths to the Best Solution", *Computer Design*, 27, n. 4 (15 febbraio 1988), 59-74.

Altri bus e prodotti

Borril, Paul L., "A Comparison od 32-Bit Buses", *IEEE MIcro*, 5, n. 6 (dicembre 1985), 71-78.

Falk, Howard, "Software Opens Door to MS-DOS Applications for Non PCs", *Computer Design*, 27, n. 2 (15 gennaio 1988), 22-25.

Marshall, Trevor, Christofer Jones, e Sigi Kluger, "The Definicon 68020 Coprocessor, part II: Software Support", *Byte*, 11, n. 8 (agosto 1986), 108-114.

Wilcox, David C., "Just How Powerful Is a Definizon Coprocessor Based System?" *Definicon Bulletin*, 1, n. 2 (autunno 1987), 15- 16.

CAPITOLO 15

La fonte di informazioni complete sull'MC68030 è l'*MC68030 User's Manual*, pubblicato dalla Motorola, Inc. Questo manuale contiene una descrizione dettagliata del chip ed un certo numero di esempi di applicazioni. ALcuni altri riferimenti sono inclui nella lista qui per l'MC68030 e l'MC68882. Le riviste più note di computer (*Byte, IEEE Computer*, ecc.) presentano spesso articoli concernenti la famiglia di prodotti della Motorola.

L'articolo di Thomas Johnson descrive l'MC68030 e discute la sua architettura. L'articolo di Lieberman evidenzia le prestazioni dell'MC68030. Varie architetture sono descritte da Marrin in un articolo che presenta anche i vantaggi e gli svantaggi di una MMU sul chip. Ruhland presenta un progetto hardware che mostra le relazioni delle linee di segnale dell'MC68020/MC68851 con quelle dell'MC68030. Il progetto del sistema globale è il soggetto dell'articolo di Wilson, che include esempi basati sull'MC68030.

Il coprocessore MC68882 è descritto in dettaglio nella ristampa dell'articolo AR244) di Beims. Questi confronta l'MC68881 con l'MC68882 e fornisce esempi di programmazione che traggono vantaggio dalle operazioni concorrenti (in parallelo) della CPU e dell'MC68882. L'articolo di Thompson fornisce una breve valutazione dell'MC68882, confrontando le sue prestazioni con quelle dell'MC68881 per vari programmi di benchmark.

Un capitolo nel libro di testo di Slater descrive la maggior parte dei tipi più noti di memoria. Sono presentate varie modalità operative speciali della memoria, volte a ridurne il tempo di accesso, che sarebbero adatte a permettere l'accesso nel modo a raffica da parte dell'MC68030.

MC68030

Johnson, Thomas L., "The RISC/CISC Melting Pot", *Byte*, 12, n. 4 (aprile 1987), 153-160.

Lieberman, David, "The 68030 Microprocessor: A Window on 1988 Computing", *Computer Design*, 27, n. 1 (1 gennaio 1988), 20-24.

Marrin, Ken, "Mainframe Architectures and Compiler Techniques Power 32-Bit Micros", *Computer Design*, 26, n. 3 (1 febbraio 1987), 57-75.

Ruhland, Michael, "PLDs and 68020 Provide Launchpad for 68030 Designs", *ESD*, 17, n. 8 (agosto 1987), 45-52.

Wilson, Ron, "Designers Seek New Approaches to Open I/O Bottlenecks", *Computer Design*, n. 2 (15 gennaio 1988), 57-73.

MC68882

Beims, Bob, *The Floating-Point Performance Standard Gets Even Faster*, Motorola Publication AR244. Phoenix, Arizona: Motorola, Inc., 1986.

Thompson, Tom, "Fast Math - A First Look at Motorola's 68882 Math Coprocessor", *Byte*, 12, n. 14 (dicembre 1987), 120-121.

Progettazione dell'hardware

Slater, Michael, *Microprocessor Based Design*. Mountain View, California: Mayfield Publishing Company, 1987.

INDICE DELLE ISTRUZIONI

Questo indice include tutte le istruzioni dell'MC68020. Il lettore può consultare l'app. C per il formato in linguaggio assembler delle istruzioni. L'app. D descrive in dettaglio ciascuna istruzione. L'insieme di istruzioni del linguaggio assembler dell'MC68851 e dell'MC68881/2 è presentato nell'app. C.

A

B

C

D

E

I

J

L

M

N

O

P

R

S

T

U

INDICE ANALITICO

Per ulteriori informazioni, si rimanda il lettore alle "Descrizioni dei capitoli", al "Sommario degli argomenti principali" ed alla "Lista dei programmi per argomento", nella prefazione del libro. Le istruzioni specifiche dell'MC68020 e dell'MC68030 sono elencate nell'indice delle istruzioni.

A

N

O

P

T

68020
68030

PROGRAMMAZIONE INTERFACCIAMENTO E PROGETTAZIONE

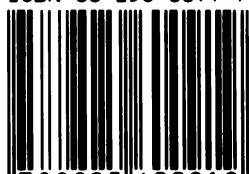
THOMAS L. HARMAN

Il 68020 ed il 68030 sono i principali dispositivi della nuova famiglia di microprocessori a 32 bit prodotti dalla Motorola. Ciascuno di essi è un microprocessore su singolo chip progettato per operare come unità centrale di elaborazione di un sistema avanzato di computer. Le loro caratteristiche ed i loro impieghi sono trattati in dettaglio in questo libro, sia per quanto riguarda la programmazione in linguaggio assembler, sia per la progettazione dell'interfaccia e del sistema.
Il libro può servire come fonte di consultazione, in quanto gli argomenti sono organizzati in base alla loro funzione ed all' importanza che rivestono per il progetto di programmi, di interfacce o di sistemi.
Il libro è organizzato in quattro parti. Nella prima viene presentata al lettore la famiglia del 68020 con una più generale introduzione ai microcomputer ed all'aritmetica dei computer. Nella seconda parte vengono analizzate le tecniche di programmazione in linguaggio assembler e la progettazione e lo sviluppo di sistemi per computer basati sul 68020. Nella terza parte sono trattati gli aspetti dell' hardware del 68020, incluso il VMEbus. Nell'ultima parte è descritto in modo ampio il processore 68030.
Prima delle appendici, il lettore potrà trovare le risposte alla maggior parte dei problemi presentati nel testo. Le appendici costituiscono un utile riepilogo per il programmatore, in quanto comprendono il linguaggio assembler ed il linguaggio macchina per la famiglia del 68020.

● Caratteristiche generali dei microprocessori ● Linguaggio assembler e istruzioni fondamentali del 68020 ● Tecniche di programmazione ● Applicazioni avanzate del 68020 ● VME bus e relativi sistemi ● Caratteristiche ed istruzioni del 68030

GRUPPO EDITORIALE JACKSON

Prentice Hall International

ISBN 88-256-0091-7

9 788825 600919

L. 85.000 Cod. GE866

844

THOMAS L. HARMAN

68020
68030

PROGRAMMAZIONE INTERFACCIAMENTO
E PROGETTAZIONE